LAMP: Linux, Apache, MySQL y PHP/Perl

Diego Chaparro (*dchaparro@acm.org*)
5 de Abril de 2006

Preámbulo

Este documento no pretende ser una guía completa de todas las tecnologías explicadas, sino simplemente una pequeña guía basada en notas que sirva como introducción a cada uno de los entornos aquí descritos.

Este documento no está libre de errores y está en continuo desarrollo.

Licencia

Índice de contenido

1 Introducción

1.1 GNU/Linux

- Sistema operativo basado en **Software Libre**
- **Libre ≠ Gratis**
- El software libre proporciona 4 libertades:
  - Libertad de **uso**
  - Libertad de **redistribución**
  - Libertad de **modificación**
  - Libertad de **redistribución de las modificaciones**
- No comparte origen ni diseño con ningún Unix, pero sí su filosofía de diseño (**Minix**).
- Inicialmente creado por **Linus Torvalds en 1991**, proyecto de tesis
- Pero linux es solo el kernel. **GNU** es el conjunto de aplicaciones
- **Distribución**: recopilación de un kernel de Linux y un conjunto de aplicaciones disponibles en el mundo del Software Libre.
- Ejemplos: Debian, Red-Hat (Fedora), Mandriva, Suse, Ubuntu, ...

1.2 Apache

- Potente y flexible servidor web
- Usado en más del 68% de todos los servidores web (datos de netcraft en Marzo 2006 [6])
- Altamente configurables y extensibles a través de módulos
- Versiones para Windows, Netware, OS/2, Unix, Linux, ...

1.3 PHP

Características generales:

- **No** es un lenguaje de **propósito general**
- Es un lenguaje "**Open Source**"
- **Interpretado** de **alto nivel**
- **Embebido** en páginas HTML.
- **Ejecutado en el servidor**
- **Compatible** con varios **sistemas operativos** (Linux, Unix, Windows, Mac OS, ...) y **servidores web** (Apache, IIS, Netscape, ...)
- Programación **procedimental** o programación **orientada a objetos**
- Compatible con gran número de **bases de datos**: dBase, Informix, MySQL, Oracle, PostgreSQL, Sybase, ODBC, Interbase, Adabas D, ...
- Soporte para muchos protocolos: LDAP, IMAP, SNMP, NNTP, POP3, HTTP y muchos otros
- Y muchas cosas más ;-)
- Más información en [5]

1.4 Perl

Características generales:

- Lenguaje de **propósito general**
- Originalmente desarrollado para **tratamiento de textos**, pero actualmente utilizado para muchos propósitos: **administración de sistemas**, **desarrollo web**, **programación de red**, ...
- Su diseño se basa en ser un **lenguaje práctico**: fácil de usar, eficiente y completo.

- Lenguaje **interpretado**
- Programación **procedimental** o programación **orientada a objetos**
- Impresionante colección de **módulos perl** (**CPAN**): ficheros, protocolos, bases de datos, gráficos, seguridad, ...
- Más información en [4]

1.5 MySQL

- Servidor de bases de datos relacionales “Open source”
- Rápido, seguro, y fácil de usar.
- Amplio subconjunto del lenguaje SQL
- Accesible desde un gran número de lenguajes de programación
- Replicación

1.6 Herramientas adicionales

1.6.1 CVS

- Implementa un sistema de control de versiones
- Permite que varios desarrolladores colaboren
- El proyecto completo está en el servidor (repositorio) y cada cliente tiene una copia
- Cada cliente modifica lo que necesite y sube los cambios al servidor
- Si hay conflicto el sistema avisa al usuario
- Se puede recuperar la historia
- Se pueden crear ramas de desarrollo

1.6.2 SSH

- Protocolo para acceso a máquinas remotas
- Ejecución de comandos en la máquina remota
- Normalmente acceso en modo texto
- Ventaja principal frente a telnet o rlogin: cifrado
- También se utiliza para copiar ficheros mediante canales cifrados: scp o sftp.
- Cliente para windows: putty, pscp, psftp.

2 Linux

2.1 Sistema de ficheros

- En forma de árbol
- El directorio raíz se expresa con el símbolo **/**
- PATH relativo: ruta entre el directorio actual y un fichero o directorio
- PATH absoluto: ruta entre el directorio raíz y un fichero o directorio
- Los nombres de ficheros y directorios son sensibles a mayúsculas y minúsculas
- Los nombres de ficheros no deberían contener espacios en blanco ni caracteres “extraños”

2.2 Comandos básicos

- **ls**: muestra el contenido de un directorio. Ejemplo:

  ls /tmp/

- **cd**: Cambiar de directorio. Ejemplo:

  cd /tmp/

- **pwd**: Muestra el directorio actual. Ejemplo:

  pwd

- **mkdir**: Crear directorios. Ejemplo:

  mkdir /tmp/dir1

- **rm**: Borra ficheros o directorios. Ejemplo:

  rm /tmp/prueba.txt

- **cp**: Copia ficheros o directorios. Ejemplo:

  cp /usr/prueba.txt /tmp/

- **mv**: Mover ficheros o directorios. Ejemplo:

  mv /usr/prueba2.txxt /tmp/

- **less**: Muestra el contenido de un fichero. Ejemplo:

  less /tmp/prueba2.txt

3 Apache

3.1 HTTP

- Protocolo de transferencia de Hipertexto
- Usado en cada transacción de la web
- Es un protocolo sin estado
- Para mantener estado se utilizan las cookies
- La versión actual es HTTP 1.1 [7]

3.1.1 GET

- Método para obtener un recurso de un servidor
- Para obtener un recurso con el URL http://host.com/index.html

1. Se abre un socket con el host host.com, puerto 80 que es el puerto por defecto para HTTP.
2. Se envía un mensaje en el estilo siguiente:

```
GET /index.html HTTP/1.0
From: yo@miHost.example
User-Agent: HTTPTool/1.0
[Línea en blanco]
```

3. La respuesta del servidor está formada por encabezados seguidos del recurso solicitado, en el caso de una página web:

```
HTTP/1.0 200 OK
Date: Fri, 31 Dec 2003 23:59:59 GMT
Content-Type: text/html
Content-Length: 1221

<html>
  <body>
    <h1>Página principal de tuHost</h1>
    (Contenido)
    .....
  </body>
</html>
```

- Si se utiliza en un formulario, los valores de los campos van en la URL

3.1.2 POST

- Es el método adecuado para los formularios que no solo sirven para obtener información
- Los datos del formulario van en el cuerpo del mensaje

3.1.3 Códigos de respuesta

- Son códigos de respuesta del servidor al cliente
- Son códigos de tres cifras:
  - 1xx Mensajes de información
  - 2xx Operación con éxito
  - 3xx Redirección hacia otra URL
  - 4xx Error por parte del cliente
  - 5xx Error por parte del servidor

4 PHP

4.1 Referencia del lenguaje

4.1.1 Sintaxis

- El código php está delimitado por unas marcas de principio y fin, que normalmente suelen ser las siguientes:

```
<?php
        CODIGO_PHP
?>
```

- Cada instrucción debe ser terminada por el símbolo **;**
- Hay varias formas de escribir comentarios en el código php:

```
# Comentario
// Comentario
/* Comentario */
```

4.1.2 Tipos de datos

- PHP no requiere la definición explícita de tipos en la declaración de variables
- El tipo de una variable es determinado mediante el contexto.
- El tipo puede cambiar durante la ejecución del programa
- PHP Soporta 8 tipos de datos primitivos:
  - Cuatro escalares:
    - **boolean**: dos posibles valores:
      - FALSE, 0, 0.0, “”, ...
      - TRUE, 1, 2, 3, “ddd”, ...
    - **integer**: número sin decimales (en decimal, octal (0+n) o hexadecimal (0xn))
    - **float**: números de coma flotante (1.234)
    - **string**: serie de caracteres entre comillas simples o comillas dobles.
  - Dos tipos compuestos:
    - **array**: conjunto de valores. Puede ser una matriz real, un vector, una tabla, ... Ejemplos:

```
array (5 => 1, 12 => 2);
array (5,3,2,1,0);


foreach ($matriz as $i => $valor) {
        print $i;
}
```

    - **object**: objeto de una clase (POO). Ejemplo:

```
new foo;
```

- Y dos tipos especiales:
  - **resource**: variable especial, que contiene una referencia a un recurso externo como ficheros, conexión a bases de datos, ...
  - **null**: representa una variable que no tiene valor.

- Se puede convertir una variable a otro tipo de esta forma:

(int) $var	Convierte la variable *var* a tipo entero
(string) $var	Convierte la variable *var* a tipo cadena

4.1.3 Variables

- Se representan con un símbolo dólar delante del nombre.
- El nombre es **sensible a mayúsculas y minúsculas**
- El nombre debe empezar con una letra o subrayado, seguido de cualquier número de letras, subrayados o números
- Dos tipos de asignación:
  - Asignación por valor: *$a = $b;*
  - Asignación por referencia: *$a = & $b;*
- Las variables tienen ámbito local dentro del contexto en el que están definidas. Si quiero utilizar una variable global dentro de un ámbito local debe expresarse que esa variable es global, en otro caso se supone que es local, de esta forma:

  global $a; // La variable está definida en un contexto superior.

- Variable *static*: variable de ámbito local en una función pero que no pierde su valor entre las llamadas a la función.
- Acceso a variables externas. Procedentes de las operaciones GET y POST

4.1.4 Expresiones

- La forma más simple y ajustada de definir una expresión es "**cualquier cosa que tiene un valor**".
- Ejemplos:

```
5
5 + 2
$a++
++$a
```

4.1.5 Operadores

Estos son los operadores más habituales:

- Aritméticos: + - * / %
- Asignación: = += -= .=
- Comparación: == != <> < > <= >=
- Incremento/Decremento: ++ --
- Lógicos: and or xor ! && ||
- De cadenas: .

4.1.6 Estructuras de control

Las estructuras de control más usadas son las siguientes:

- **if**:

```
if (expr)
        sentencia
```

- **if – else**:

```
if (expr) {
        sentencia
} else {
        sentencia
}
```

- **if-elseif**:

```
if (expr) {
        sentencia
} elseif ($a == $b) {
        sentencia
} else {
        sentencia
}
```

- **Sintaxis alternativa**:

```
if (expr):
        sentencia
        sentencia
elseif (expr):
        sentencia
        sentencia
else:
        sentencia
endif
```

- **while**:

```
while (expr) sentencia

while (expr) {
        sentencia
}
```

- **do-while**:

```
do {
        sentencia
} while (expr);
```

- **for**:

```
for (expr1; expr2; expr3) {
        sentencia
}
```

expr1 se evalúa al principio del bucle
expr2 se evalúa al comienzo de cada iteración, si es TRUE el bucle continúa y sino el bucle finaliza
expr3 se evalúa al final de cada iteración

- **foreach**:

  foreach (expresion_array as $value) sentencia
  foreach (expresion_array as $key => $value) sentencia

- **return**:

  Cuando se ejecuta desde una función termina inmediatamente la ejecución de la función y devuelve su argumento como valor de la función

- **include** y **require**:

  Se utilizan para incluir y evaluar el archivo especificado.

4.1.7 Funciones

- No es necesario definirlas antes de referenciarlas
- Pasar argumentos por valor:

  function takes_array ($input)

- Pasar argumentos por referencia:

  function takes_array (& $input)

- Argumentos con valor por defecto:

  function takes_array ($input = “foo”)

- Devolver valor de retorno:

  return ($resultado)

4.1.8 Clases y objetos

- Una clase es una colección de variables y funciones que trabajan con éstas variables

- Definición de una clase:

```
class Contador {
        var $numero;

        function incrementar (){
                $this->numero += 1;
        }
}
```

- Definición de un objeto:

```
$objeto = new Contador;
```

- Uso de un objeto:

```
$objeto->incrementar();
```

- **Herencia**: puedo extender las funcionalidades de una clase:

```
class Contador_dec extends Contador {

        function decrementar (){
                $this->numero -= 1;
        }
}
```

- **Constructor**: son funciones en una clase que son llamadas automáticamente cuando se crea una nueva instancia de una clase con **new**

```
class Contador_init extends Contador {

        function Contador_init ( $num ){
                $this->numero = $num;
        }
}
```

4.1.9 Excepciones

En PHP4 no hay control de excepciones :-(

Pero si está incluido en PHP5 :-)

4.2 Más características

4.2.1 Autenticación HTTP con PHP

- Se puede enviar un mensaje de "Autenticación requerida" al navegador utilizando la función header con las siguientes opciones:

```
header('WWW-Authenticate: Basic realm="My Realm"');
header('HTTP/1.0 401 Unauthorized');
```

- Ejemplo real:

```
<?php
   if (!isset($_SERVER['PHP_AUTH_USER'])) {
       header('WWW-Authenticate: Basic realm="My Realm"');
       header('HTTP/1.0 401 Unauthorized');
       echo 'Text to send if user hits Cancel button';
       exit;
   } else {
       echo "<p>Hello {$_SERVER['PHP_AUTH_USER']}.</p>";
       echo "<p>You entered {$_SERVER['PHP_AUTH_PW']} as
             your password.</p>";
   }
?>
```

4.2.2 Cookies

- Mecanismo que sirve para almacenar datos en el navegador del usuario remoto, para así poder identificar al usuario cuando vuelva

- Las cookies son parte de la cabecera HTTP, por tanto la función setcookie() debe ser llamada **antes de que se produzca cualquier salida al navegador**

- Se pueden poner cookies usando la función setcookie()

- Ejemplo para establecer una cookie:

```
<?php
        $valor = 'valor de la cookie';
        setcookie("CookieDePrueba", $valor);
        /* o fijar que expire en 1 hora */
        setcookie("CookieDePrueba", $valor, time()+3600);
?>
```

- Obtener el valor de una cookie del cliente:

```
<?php
      //Imprime una cookie individual
      echo $_COOKIE["CookieDePrueba"];
      echo $HTTP_COOKIE_VARS["CookieDePrueba"];
      // Para ver todas las cookies
      print_r($_COOKIE);
?>
```

4.2.3 Funciones de fecha y hora

Estas son algunas de las funciones de fecha y hora más utilizadas:

- **date** ([formato [, marca_tiempo]]): Dar formato a una fecha/hora local. Se pueden encontrar todas las opciones de formato en [8]. Ejemplos:

```
// Imprime algo como: Monday 15th of August 2005 03:12:46 PM
echo date('l dS \of F Y h:i:s A');
// Imprime: July 1, 2000 is on a Saturday
echo "July 1, 2000 is on a " . date("l", mktime(0, 0, 0, 7, 1, 2000));
```

- **getdate** ([marca_tiempo]): devuelve un array con los valores de fecha y hora actuales. Ejemplo:

```
$hoy = getdate();
print_r($hoy);
```

- **mktime** ([int hora [, int minuto [, int segundo [, int mes [, int dia [, int anyo]]]]]]): obtiene la marca de tiempo de los datos especificados. Ejemplo:

```
echo date ("M-d-Y", mktime(0, 0, 0, 12, 32, 1997))
```

4.3 Editores

Hay muchos editores de texto y Entornos Integrados de Desarrollo que se pueden usar para crear, editar, y organizar archivos PHP. Contar con un editor que resalte la sintaxis de PHP puede ser de mucha ayuda. Hay una lista parcial de éstos en [3].

Pero en realidad cualquier editor de texto es suficiente para poder hacerlo.

4.4 Conexión con MySQL

- Conectarse a MySQL:

  $link= mysql_connect(“localhost”, “usuario”, “password”);

- Seleccionar la base de datos:

  $bbdd= mysql_select_db('nombre_bbdd', $link) or die('Error');

- Realizar operación:

  ```
  $query = 'SELECT * FROM mi_tabla';
  $resultado = mysql_query($query, $bbdd) or die('Error: ' .
          mysql_error());
  ```

- Liberar el conjunto de resultados:

  mysql_free_result($resultado);

- Cerrar la conexión:

  mysql_close($link);

5 Perl

5.1 Referencia del lenguaje

5.1.1 Sintaxis

- Las instrucciones terminan en el carácter **;**
- Los comentarios se indican con el carácter **#**
- Las cadenas de texto se especifican entre **comillas dobles o simples** (aunque las variables solo se interpretan entre las dobles)
- Al llamar a una función, los paréntesis entre los argumentos son opcionales

5.1.2 Tipos de datos

- Escalares:
  - **Cadenas de texto**:

    "*camel*"

  - **Enteros**:

    56

  - **Float**:

    7.53

- **Arrays**: es un listado de valores

  ("camel", "llama", "owl")
  (42, “valor”)

- **Hash o listas asociativas**: listado de valores asociados con una clave:

  ("apple", "red", "banana", "yellow")

  (apple => "red", banana => "yellow",)

5.1.3 Variables

- Las variables tipo escalar se identifican por el prefijo **$**
- Las variables de tipo array se identifican por el prefijo **@**
- Las variables del tipo hash se identifican por el prefijo **%**
- El nombre es sensible a mayúsculas y minúsculas
- El nombre debe empezar con una letra, seguido de cualquier número de letras, subrayados o números
- No es necesario definir el tipo al definir una variable
- El **ámbito de una variable** se define así:
  - Variable **local**: *my $var = "valor";*
  - Variable **global**: *$var = "valor";*
- El 0 es el primer índice de un array
- *$#var* muestra el índice del último elemento de un array

5.1.4 Operadores

Estos son los operadores más habituales:

- Aritméticos: + - * / %
- Asignación: = += -= .=
- Comparación: == != < > <= >=
- Comparación de cadenas: eq, ne, lt, gt, le, ge
- Incremento/Decremento: ++ --

- Lógicos: *not and or ! && ||*
- Otros: *.* (concatenar) *..* (rangos)

5.1.5 Estructuras de control

Las estructuras de control más usadas son las siguientes:

- **if**:

```
if ( condition ) {
        ...
} elsif ( other condition ) {
        ...
} else {
        ...
}
```

También en la versión opuesta:

```
unless ( condition ) {
        ...
}
```

- **while**:

```
while ( condition ) {
        ...
}

until ( condition ) {
        ...
}
```

También como post-condición:

```
print "LA LA LA\n" while 1;
```

- **for**:

```
for ($i=0; $i <= $max; $i++) {
        ...
}
```

- **foreach**:

```
foreach my $key ( keys %hash) {
        print "The value of $key is $hash{$key}\n";
}
```

5.1.6 Funciones

- Definición:

```
sub nombre_de_función {
        instrucciones;
        [return Variable o expresión;]
}
```

- Obtener los argumentos de la llamada a la función:

```
my ($key, $value) = @_;
```

o bien:

```
$key = @_[0];
$value = @_[1];
```

- Argumentos por referencia:

```
my (*key, *value) = @_;
```

5.2 Más características

5.2.1 Expresiones regulares

- **Expresiones regulares de comparación**: Nos permiten evaluar si un patrón de búsqueda se encuentra en una cadena de caracteres, de modo que mediante este tipo de expresiones regulares obtendremos un valor lógico verdadero o falso según se encuentre el patrón deseado. Sintaxis:

  valor a comparar =~ patrón de búsqueda

  Ejemplo:

```
if ($linea =~ /html/) {
        ...
}
```

- **Expresiones regulares de sustitución**: Las expresiones regulares de sustitución permiten cambiar los patrones de búsqueda por caracteres nuevos definidos por el usuario que componen el patrón de sustitución. Sintaxis:

  variable =~ s/patrón de búsqueda/patrón de sustitución/opciones

  Ejemplos:

  ```
  $a =~ s/foo/bar/;       # replaces foo with bar in $a

  $a =~ s/foo/bar/g;      # replaces ALL INSTANCES of foo
                            with bar in $a
  ```

- **Expresiones regulares de traducción**: En este caso se trata de comparar uno a uno los caracteres del patrón de búsqueda con los de la cadena de sustitución, de modo que cada vez que se encuentra una ocurrencia que coincide con uno de los caracteres del patrón se intercambia por su correspondiente en la cadena del patrón de sustitución. Sintaxis:

  variable =~ tr/patrón de búsqueda/cadena a traducir/opciones

  Ejemplos:

  ```
  $var =~ tr/A-Z/a-z/; # transforma mayúsculas a minúsculas

  $cnt = $var =~ tr/*/*/; # cuenta los arteriscos de $var
  ```

5.2.2 Funciones relacionadas con cadenas

- **length**(cadena de caracteres). Esta función nos permite conocer la longitud de una cadena de caracteres. Por ejemplo:

- **chop**(cadena de caracteres). Elimina el último carácter de la ristra y retorna dicho carácter. Esta función se suele usar para eliminar el carácter de nueva línea que contienen las ristras que se introducen por teclado o se leen de un fichero.

- **index**(cadena, subcadena, [posición]). Esta función retorna la posición de la primera ocurrencia de la subcadena en la cadena indicada. El parámetro posición indica el número de caracteres desde el inicio que se deben ignorar en la búsqueda.

- **rindex**(cadena, subcadena, [posición]). Esta posición trabaja igual que index salvo que retorna de la última ocurrencia de la subcadena en la cadena. Posición es el número de caracteres desde el inicio que se ignorá en la búsqueda.

- **substr**(cadena, desplazamiento, [longitud]). Esta función extrae una subcadena de la cadena dada, desde la posición indicada por desplazamiento hasta el número de caracteres indicado por longitud.

- **split**(caracter, cadena). Separa en trozos una *cadena* y la convierte en un array. Separa la cadena en base al delimitador *carácter* seleccionado.

5.2.3 Funciones relacionadas con listas asociativas

- **keys:** El operador keys proporciona la lista de claves o índice de una lista asociativa. Por ejemplo:

```
%cuotas = ("root", 10000, "pat", 256);
@list = keys(%cuotas);          # @list = ("root", "pat")
```

- **values:** El operador values devuelve los valores de una lista asociativa. Por ejemplo:

```
%cuotas = ("root", 10000, "pat", 256);
@list = values(%cuotas);        # @list = (10000, 256)
```

- **each:** También se puede acceder a una lista asociativa por pares clave-valor, permitiendo el operador each recorrerla iterativamente. El ejemplo siguiente ilustra un acceso a los pares clave-valor una lista asociativa.

```
%cuotas = ("root", 10000, "pat", 256);
while (($clave, $valor)=each(%cuotas)) {
                print "Login: $clave, Cuota: $valor\n";
}
```

- **delete:** Para suprimer elementos de una lista asociada se usa el operador delete. Este operador permite suprimir un par clave-valor de una lista asociativa. Por ejemplo:

```
%cuotas = ("root", 10000, "pat", 256);
delete $cuota{"pat"};           #%cuotas = ("root", 10000)
```

5.2.4 Entrada/salida

- **Punteros predefinidos**:

  STDIN, STDOUT, STDERR

- **Acceso a ficheros**:

  open (PUNTERO, "modo de acceso + nombre de archivo");

  El modo de acceso puede ser:

<	Solo lectura (por defecto)
>	Escritura
>>	Escritura al final del fichero
+>	Lectura/escritura

- **Escribir en un fichero**:

  print PUNTERO “Texto...”

- **Leer de un fichero**:

```
open(PASSWD, "/etc/passwd");
while ($p = <PASSWD>) {    # lee una línea del archivo
        chop($p);          #quitamos el salto de línea
        @field = split(/:/, $p);
        print " Usuario $field[0] y su directorio es $field[5]";
}
close(PASSWD);
```

- **Obtener una línea de la entrada estándar**:

```
$uno = <STDIN>;
```

5.2.5 Funciones de fecha y hora

- Hay varios módulos que se encargan de realizar operaciones con fechas y horas en perl, pero hay una función incluida que podemos utilizar para las operaciones básicas:

  localtime()

- Esta función devuelve un array con los siguientes valores:
  - Segundos
  - Minutos
  - Hora
  - Día
  - Mes
  - Año desde el 1900
  - Día de la semana
  - Número de día del año
- Ejemplo:

```
($sec, $min, $hour, $mday, $mon, $year,  $wday, $yday,
        $resto)=localtime(time);
printf "%4d-%02d-%02d %02d:%02d:%02d\n", $year+1900,
        $mon+1, $mday, $hour, $min, $sec;
```

5.3 Conexión con MySQL

- Cargar el módulo de acceso a bases de datos:

```
use DBI();
```

- Crear la conexión a la base de datos:

```
$bbdd = DBI->connect ( “DBI:mysql:database=test; host=localhost”,
                “prueba”,
                “prueba”,
                {'RaiseError' => 1}
        );
```

- Realizar la operación:

```
$query = “SELECT * FROM tabla”;
$order = $bbdd->prepare ($query);
$order->execute;
```

- Liberar resultados:

```
$order->finish;
```

- Cerrar la conexión:

```
$bbdd->disconnect();
```

6 MySQL

6.1 Permisos de acceso

Para cada usuario que se conecta se verifican varias cosas:

- La máquina desde la que se conecta. El acceso puede restringirse para cada usuario desde cada máquina.
- Para cada operación que quiere realizar el usuario se verifica si tiene los suficientes permisos para realizarla.

Esta información de usuarios, accesos y permisos se almacenan en las tablas de la base de datos *mysql*.

6.2 Acceso de usuarios

Para acceder a la base de datos se puede hacer de la siguiente forma:

```
shell> mysql -u monty -p db_name
shell> mysql -u monty -pguess db_name
```

6.3 Creación de usuarios

Primero, conectarse como administrador de la base de datos:

```
shell> mysql -u monty -pguess db_name
```

Después,

```
mysql> GRANT ALL PRIVILEGES ON *.* TO
'monty'@'localhost' IDENTIFIED BY 'some_pass' WITH GRANT
OPTION;
```

NOTA 1: El *.* significa BBDD.tabla

NOTA 2: Para especificar desde cualquier máquina: 'monty'@'%'

NOTA 3: "ALL PRIVILEGES" puede ser substituido por un conjunto de los siguientes separados por comas: SELECT, INSERT, UPDATE, DELETE, CREATE, DROP, ...

Otro modo de hacerlo sería:

```
shell> mysql -u root mysql
mysql> INSERT INTO user VALUES ('localhost', 'monty',
PASSWORD('some_pass'), 'Y', 'Y', 'Y', 'Y', 'Y', 'Y',
'Y',    'Y', 'Y', 'Y', 'Y', 'Y', 'Y', 'Y');
mysql> FLUSH PRIVILEGES;
```

6.4 Restaurar contraseña del administrador

1. Crear un fichero de texto que contenga lo siguiente:

```
SET PASSWORD FOR 'root'@'localhost' = PASSWORD('newpass');
```

Guarda el fichero y asígnale un nombre, por ejemplo ~/mysql-init

2. Arranca el servidor MYSQL con la opción --init-file=~/mysql-init:

```
shell> mysqld_safe --init-file=~/mysql-init &
```

El contenido del fichero es ejecutado al arrancar el servidor cambiando la contraseña del usuario root.

6.5 Operaciones habituales

Mostrar bases de datos existentes:

```
mysql> show databases;
```

Conectarse a una base de datos:

```
mysql> use bd_prueba;
```

Mostrar las tablas que hay en la base de datos:

mysql> show tables;

Mostrar la estructura de una tabla:

mysql> desc nombre_tabla;

Crear una tabla:

mysql> CREATE TABLE agenda (id INT NOT NULL, nombre CHAR(50), direccion CHAR(100) , KEY (id))

Crear una tabla con claves ajenas:

mysql> CREATE TABLE email (id INT NOT NULL, email CHAR(50), agenda_id INT, FOREIGN KEY (agenda_id) REFERENCES agenda(id))

6.6 Tipos de datos

6.6.1 Números

Algunos de ellos son:

- **TINYINT**[(M)] [UNSIGNED] [ZEROFILL]

  Un entero muy pequeño. Su rango es de -128 a 127. Para enteros sin signo el rango es de 0 a 255.

- **INT**[(M)] [UNSIGNED] [ZEROFILL]

  Un entero normal. Su rango es de -2147483648 a 2147483647. Para enteros sin signo el rango es de 0 a 4294967295.

- **FLOAT**[(M,D)] [UNSIGNED] [ZEROFILL]

  Un número pequeño de punto flotante (o bien, de precisión sencilla). El rango de valores permitidos es de -3.402823466E+38 a -1.175494351E-38, 0, y de 1.175494351E-38 a 3.402823466E+38.

6.6.2 Fecha y hora

- **DATE**

  Valores de fecha. “YYYY-MM-DD”

- **TIME**

  Valores de hora. “HH:MM:SS”

- **DATETIME**

  Valores de fecha y hora. “YYYY-MM-DD HH:MM:SS”

6.6.3 Cadenas de caracteres

- **CHAR** (n)

  Cadena de caracteres de tamaño *n*

- **TINYTEXT**

  Una columna TEXT con una longitud máxima de 255 caracteres.

7 Herramientas adicionales

7.1 CVS

- Para usar autenticación con ssh:

  export CVSROOT=":ext:USUARIO@www.url.com:/home/cvs"
  export CVS_RSH=/usr/bin/ssh

- Para usar autenticación con pserver:

  export CVSROOT=":pserver:USUARIO@www.url.com:/home/cvs"

- Bajar por primera vez el módulo:

  cvs co [nombre-del-módulo]

- Actualizar cambios:

  cd [directorio-del-módulo]
  cvs update

- Publicar nuestras modificaciones:

  cvs commit

- Resolución de conflictos:

  [...]
  >>>>>>>>>>>>>>>
  taca
  ==========
  tacataca
  <<<<<<<<<<<<<<<
  [...]

- Añadir ficheros:

  cvs add [fichero]

- Eliminar ficheros:

  cvs remove [fichero]

- Clientes gráficos:
  - tkcvs
  - wincvs

7.2 SSH

- Conectarse a una máquina:

  ssh usuario@maquina

- Copiar un fichero a una máquina:

  scp path_fichero usuario@maquina:path_destino

- Copiar un fichero desde una máquina remota:

  scp usuario@maquina:path_destino path_fichero

8 Referencias

[1] http://www.mysql.com

[2] http://www.mysql-hispano.org/

[3] http://www.thelinuxconsultancy.co.uk/phpeditors.php

[4] http://perldoc.perl.org

[5] http://www.php.net/docs.php

[6] http://news.netcraft.com/archives/web_server_survey.html

[7] http://www.ietf.org/rfc/rfc2616.txt

[8] http://www.php.net/manual/es/function.date.php