Administración de datos

Remo Suppi Boldrito

PID_00174428

Universitat Oberta de Catalunya
www.uoc.edu

Índice

Introducción 5

Objetivos 6

1. Administración de datos 7
1.1. PostgreSQL 8
1.1.1. Instalación de PostgreSQL 8
1.1.2. ¿Cómo se debe crear una DB? 10
1.1.3. ¿Cómo se puede acceder a una DB? 11
1.1.4. Usuarios de DB 11
1.1.5. Mantenimiento 13
1.1.6. Resumen de la instalación de PostgreSQL 14
1.2. El lenguaje SQL 14
1.2.1. Entornos de administración gráficos 16
1.3. MySQL 16
1.3.1. Instalación de MySQL 17
1.3.2. Postinstalación y verificación de MySQL 18
1.3.3. El programa monitor (cliente) mysql 19
1.3.4. Caso de uso: procesamiento de datos con MySQL 20
1.3.5. Administración de MySQL 23
1.3.6. Interfaces gráficas 23
1.4. Source Code Control System (CVS y Subversion) 24
1.4.1. Revision Control System (RCS) 25
1.4.2. Concurrent Versions System (CVS) 26
1.4.3. Subversion 30
1.4.4. Git 32
1.4.5. Mantis Bug Tracker 36

Actividades 38

Bibliografía 38

Introducción

Un aspecto importante de un sistema operativo es dónde y cómo se guardan los datos. Cuando la disponibilidad de estos datos debe ser eficiente, es necesario utilizar bases de datos (*databases*, DB). Una base de datos es un conjunto estructurado de datos que pueden organizarse de manera simple y eficiente por parte un gestor de dicha base. Las bases de datos actuales se denominan relacionales, ya que los datos pueden almacenarse en diferentes tablas que facilitan su gestión y administración. Para ello, y con el fin de estandarizar el acceso a las bases de datos, se utiliza un lenguaje denominado SQL (Structured Query Language), que permite una interacción flexible, rápida e independiente de las aplicaciones a las bases de datos. En este módulo se verán los gestores más importantes de bases de datos en entornos GNU/Linux, una breve reseña a SQL, así como diferentes formas de gestionar datos y repositorios dentro de un entorno distribuido y multiusuario.

Objetivos

En los materiales didácticos de este módulo encontraréis los contenidos y las herramientas procedimentales para conseguir los objetivos siguientes:

1. Analizar las maneras de almacenar datos en forma masiva y uso eficiente.
2. Desarrollar los aspectos esenciales de bases de datos y su instalación y uso.
3. Trabajar con las técnicas de control de versiones y analizar sus ventajas.
4. Instalar y analizar las diferentes herramientas de gestión de datos y su integración para entornos de desarrollo.

1. Administración de datos

En la actualidad, la forma más utilizada para acceder a una base de datos es a través de una aplicación que ejecuta código SQL. Por ejemplo, es muy común acceder a una DB a través de una página web que contenga código PHP o Perl (los más comunes). Cuando un cliente solicita una página, se ejecuta el código PHP/Perl incrustado en la página, se accede a la DB y se genera la página con su contenido estático y el contenido extraído de la DB que posteriormente se envía al cliente. Dos de los ejemplos más actuales de bases de datos son los aportados por ProstgreSQL y MySQL, que serán objeto de nuestro análisis.

Sin embargo, cuando se trabaja en el desarrollo de un software, existen otros aspectos relacionados con los datos, como su validez y su ámbito (sobre todo si existe un conjunto de usuarios que trabajan sobre los mismos datos). Existen diversos paquetes para el control de versiones (revisiones), pero el objetivo de todos ellos es facilitar la administración de las distintas versiones de cada producto desarrollado junto a las posibles especializaciones realizadas para algún cliente específico. El control de versiones se realiza para controlar las distintas versiones del código fuente. Sin embargo, los mismos conceptos son aplicables a otros ámbitos y no solo para el código fuente sino para los documentos, imágenes, etc. Aunque un sistema de control de versiones puede realizarse de forma manual, es muy aconsejable disponer de herramientas que faciliten esta gestión (CVS, Subversion, GIT, SourceSafe, Clear Case, Darcs, Plastic SCM, RCS, etc.).

En este módulo veremos **CVS (Control Version System)**, **Subversion** y **GIT** para controlar y administrar múltiples revisiones de archivos, automatizando el almacenamiento, la lectura, la identificación y la mezcla de diferentes revisiones. Estos programas son útiles cuando un texto se revisa frecuentemente e incluye código fuente, ejecutables, bibliotecas, documentación, gráficos, artículos y otros archivos. Finalmente, también se analizará una herramienta para el seguimiento de incidencias y errores en entorno de desarrollo o de proyectos llamado Mantis.

La justificación de CVS y Subversion se puede encontrar en que CVS es uno de los paquetes tradicionales más utilizados y Subversion (también se lo conoce como svn por ser el nombre de la herramienta de línea de comandos) es un programa de control de versiones diseñado específicamente para reemplazar al popular CVS y que soluciona algunas de sus deficiencias. Una característica importante de Subversion es que, a diferencia de CVS, los archivos con versiones no tienen cada uno un número de revisión independiente. Por el

contrario, todo el repositorio tiene un único número de versión que identifica un estado común de todos los archivos del repositorio en un determinado momento.

Como penúltimo punto, y dadas sus prestaciones y utilización en grandes proyectos, se describen las características y la instalación de otro proyecto GPL, llamado GIT. GIT es un software de control de versiones diseñado por Linus Torvalds, basado en la eficiencia y la confiabilidad del mantenimiento de versiones de aplicaciones cuando estas tienen un gran número de archivos de código fuente. Finalmente, para completar un mínimo círculo de herramientas para compartir y gestionar datos, se presenta una breve descripción de **Mantis Bug Tracker**, que es una herramienta de gestión de incidencias (*bug tracker*) de código abierto. Esta aplicación está escrita en PHP y requiere una base de datos y un servidor web. Desde el proyecto se recomienda la utilización de MySQL y Apache, respectivamente.

1.1. PostgreSQL

En el lenguaje de bases de datos, PostgreSQL utiliza un modelo cliente-servidor [19]. Una sesión de PostgreSQL consiste en una serie de programas que cooperan:

1) Un proceso servidor que gestiona los archivos de la DB acepta conexiones de los clientes y realiza las acciones solicitadas por estos sobre la DB. El programa servidor es llamado en PostgreSQL *postmaster*.

2) La aplicación del cliente (*frontend*) es la que solicita las operaciones que hay que realizar en la DB y que pueden ser de lo más variadas; por ejemplo: herramientas en modo texto, gráficas, servidores de web, etc.

Generalmente, el cliente y el servidor se encuentran en diferentes *hosts* y se comunican a través de una conexión TCP/IP. El servidor puede aceptar múltiples peticiones de diferentes clientes y activar para cada nueva conexión un proceso que lo atenderá en exclusiva de un modo transparente para el usuario. Existe un conjunto de tareas que pueden ser llevadas a cabo por el usuario o por el administrador, según convenga, y que pasamos a describir a continuación.

1.1.1. Instalación de PostgreSQL

Este paso es necesario para los administradores de la DB [19]. Dentro de las funciones del administrador de DB se incluye la instalación del servidor, la inicialización y configuración, la administración de los usuarios y tareas de mantenimiento de la DB. La instalación de la base de datos se puede realizar de dos modos, a través de los binarios de la distribución, lo cual no presenta

ninguna dificultad, ya que los *scripts* de distribución realizan todos los pasos necesarios para tener la DB operativa, o a través del código fuente, que será necesario compilar e instalar. En el primer caso (*pre-built binary packages*), se pueden utilizar los gestores de paquetes o desde línea de comandos, por ejemplo, en Debian el `apt-get`. Para el segundo caso, se recomienda ir siempre al origen (o a un repositorio espejo de la distribución original). Es importante tener en cuenta que después la instalación desde el código fuente quedará fuera de la DB de software instalado y se perderán los beneficios de administración de software que presenten por ejemplo apt-cache o apt-get.

Instalación desde el código fuente paso a paso

Primero se debe obtener el software del sitio (`x.x` es la versión disponible) http://www.postgresql.org/download/ y se descomprime (`x.x.x` es la versión 9.0.1 en el momento de hacer esta revisión):

```
gunzip postgresql-x.x.x.tar.gz
tar xf postgresql-7.3.tar
```

Ahora hay que cambiarse al directorio `postgresql` y configurarlo con la instrucción `./configure`.

Se compila con `gmake`, se verifica la compilación con `gmake check` y se instala con `gmake install` (por defecto, lo hará en `/usr/local/pgsql`).

Postinstalación

Hay que inicializar las variables, en bash, sh, ksh:

```
LD_LIBRARY_PATH = /usr/local/pgsql/lib;
PATH = /usr/local/pgsql/bin:$PATH;
export LD_LIBRARY_PATH PATH;
```

o bien, en csh:

```
setenv LD_LIBRARY_PATH /usr/local/pgsql/lib;
set path = (/usr/local/pgsql/bin $path)
```

Es recomendable poner esta inicialización en los *scripts* de configuración del usuario, por ejemplo `/etc/profile` o `.bashrc` para el *bash*. Para tener acceso a los manuales, se debe inicializar la variable `MANPATH` de la misma forma:

```
MANPATH = /usr/local/pgsql/man:$MANPATH;
export MANPATH
```

Una vez instalada la DB, se deberá crear un usuario que gestione las bases de datos (es conveniente crear un usuario diferente del root para que no tenga

conexión con otros servicios de la máquina), por ejemplo, el usuario postgres utilizando el comando `useradd`. A continuación, se debe crear un área de almacenamiento para las bases de datos sobre el disco (espacio único) que será un directorio, por ejemplo `/usr/local/pgsql/data`. Para ello, ejecutad el comando `initdb -D /usr/local/pgsql/data`, conectado como el usuario creado (postgres). Es posible recibir un mensaje conforme no puede crear el directorio por falta de privilegios; por ello, se deberá crear primero el directorio y después indicarle a la DB cuál es; como root, hay que hacer, por ejemplo:

```
mkdir /usr/local/pgsql/data
chown postgres /usr/local/pgsql/data
su postgres
initdb -D /usr/local/pgsql/data
```

Iniciar el servidor (que se llama `postmaster`).

Para ello, utilizar:

```
postmaster -D /usr/local/pgsql/data
```

Para ejecutarlo en modo activo (foreground) y para ejecutarlo en modo pasivo (background), utilizar:

```
postmaster -D /usr/local/pgsql/data >logfile 2>&1 &
```

Las redirecciones se hacen para almacenar los errores del servidor. El paquete también incluye un *script* (`pg_ctl`) para no tener que conocer toda la sintaxis de *postmaster* para ejecutarlo:

```
/usr/local/pgsql/bin/pg_ctl start -l logfile -D /usr/local/pgsql/data
```

Para abortar la ejecución del servidor, se puede hacer de diferentes formas, con el `pg-ctl`, por ejemplo, o bien directamente con

```
kill -INT ´head -1 /usr/local/pgsql/data/postmaster.pid´
```

1.1.2. ¿Cómo se debe crear una DB?

La primera acción para verificar si se puede acceder al servidor de DB es crear una base de datos. El servidor PostgreSQL puede gestionar muchas DB y es recomendable utilizar una diferente para cada proyecto.

Para crear una base de datos, se utiliza el comando `createdb` desde la línea de comandos del sistema operativo. Este comando generará un mensaje `CREATE DATABASE` si todo es correcto.

Es importante tener en cuenta que para llevar a cabo esta acción, debe haber un usuario habilitado para crear una base de datos, como hemos visto en el subapartado anterior, en que existe un usuario que instala la base de datos y que tendrá permisos para crear bases de datos y crear nuevos usuarios que, a su vez, puedan crear bases de datos. Generalmente (y en Debian), este usuario

es postgres por defecto. Por ello, antes de hacer el `createdb`, se debe hacer un `su postgres` (si se es root no es necesaria ninguna palabra clave, pero si se es otro usuario, necesitaremos la palabra clave del usuario postgres) y, a continuación, se podrá realizar el `createdb`. Para crear una DB llamada *nteumdb*:

```
createdb nteumdb
```

Si la ejecución del comando da error, puede ser que no esté bien configurado el camino o que la DB esté mal instalada. Se puede intentar con el camino absoluto (`/usr/local/pgsql/bin/createdb nteumdb`), que dependerá de la instalación que se haya hecho, o consultar referencias para la solución de problemas. Otros mensajes de error serían `could not connect to server`, cuando el servidor no está arrancado, o `CREATE DATABASE: permission denied`, cuando no se tienen privilegios para crear la DB. Para eliminar la base de datos, se puede utilizar `dropdb nteumdb`.

1.1.3. ¿Cómo se puede acceder a una DB?

Una vez creada la DB, se puede acceder a ella de diversas formas:

Nota

Para poder acceder a la DB, el servidor de base de datos deberá estar en funcionamiento. Cuando se instala Postgresql, se crean los enlaces adecuados para que el servidor se inicie en el arranque del ordenador. Para más detalles, consultad el subapartado de instalación.

1) ejecutando un comando interactivo llamado `psql`, que permite editar y ejecutar comandos SQL (por ejemplo, `psql nteumdb`);

2) ejecutando una interfaz gráfica como PhpPGAdmin o alguna *suite* que tenga soporte ODBC para crear y manipular DB;

3) escribiendo una aplicación con algunos de los lenguajes soportados, como PHP, Perl o Java, entre otros (consultad *PostgreSQL Programmer's Guide*).

Por simplicidad, utilizaremos `psql` para acceder a la DB, por lo que se deberá introducir `psql nteumdb`: saldrán unos mensajes con la versión e información y un *prompt* similar a `nteumdb =>`. Se pueden ejecutar algunos de los comandos SQL siguientes:

`SELECT version();` o también `SELECT current date;`

`psql` también tienen comandos que no son SQL y comienzan por `\`, por ejemplo `\h` (enumera todos los comandos disponibles) o `\q` para terminar.

1.1.4. Usuarios de DB

Los usuarios de la DB son completamente distintos de los usuarios del sistema operativo. En algunos casos, podría ser interesante mantener una correspon-

dencia, pero no es necesario. Los usuarios son para todas las DB que controla dicho servidor, no para cada DB.

Para crear un usuario, se puede ejecutar la sentencia SQL `CREATE USER` *`nombre`* y para borrar usuarios, `DROP USER` *`nombre`*.

También se puede llamar a los programas `createuser` y `dropuser` desde la línea de comandos. Existe un usuario por defecto llamado postgres (dentro de la DB), que es el que permitirá crear los restantes (para crear nuevos usuarios si el usuario de sistema operativo con el que se administra la DB no es postgres `psql -U usuario`).

Un usuario de DB puede tener un conjunto de atributos en función de lo que puede hacer:

- **Superusuario**: este usuario no tiene ninguna restricción. Por ejemplo, podrá crear nuevos usuarios: `CREATE USER nombre CREATEUSER;`
- **Creador de DB**: tiene permiso para crear DB. Para crear un usuario de estas características, utilizad el comando: `CREATE USER nombre CREATEDB;`
- **Contraseña**: solo es necesario si por cuestiones de seguridad se desea controlar el acceso de los usuarios cuando se conecten a una DB. Para crear un usuario con contraseña (`palabra_clave` será la clave para ese usuario): `CREATE USER nombre WITH PASSWORD 'palabra_clave';`

A un usuario se le pueden cambiar los atributos utilizando el comando `ALTER USER`.

También se pueden hacer grupos de usuarios que compartan los mismos privilegios con:

```
CREATE GROUP NomGrupo;
```

Para insertar usuarios en este grupo:

```
ALTER GROUP NomGrupo ADD USER Nombre1;
```

Para borrar:

```
ALTER GROUP NomGrupo DROP USER Nombre1;
```

Ejemplo: operaciones con grupo dentro de psql

```
CREATE GROUP NomGrupo;
ALTER GROUP NomGrupo ADD USER Nom1,...; ALTER GROUP NomGrupo DROP USER
Nom1,...;
```

Cuando se crea una DB, los privilegios son para el usuario que la crea (y para el *superusuario*). Para permitir que otro usuario utilice esta DB o parte de ella, se le deben conceder privilegios. Hay diferentes tipos de privilegios, como `SELECT`, `INSERT`, `UPDATE`, `DELETE`, `RULE`, `REFERENCES`, `TRIGGER`, `CREATE`,

`TEMPORARY`, `EXECUTE`, `USAGE` y `ALL PRIVILEGES` (se deben consultar las referencias para ver su significado).

Para asignar los privilegios, se puede utilizar `GRANT UPDATE ON objeto TO usuario` donde `usuario` deberá ser un usuario válido de PostgerSQL y `objeto`, una tabla, por ejemplo.

Este comando lo deberá ejecutar el superusuario o el dueño de la tabla. El usuario PUBLIC puede ser utilizado como sinónimo de todos los usuarios y ALL como sinónimo de todos los privilegios. Por ejemplo, para quitar todos los privilegios a todos los usuarios de objeto, se puede ejecutar:

```
REVOKE ALL ON objeto FROM PUBLIC;
```

1.1.5. Mantenimiento

Hay un conjunto de tareas que son responsabilidad del administrador de DB y que se deben realizar periódicamente:

1) Recuperar el espacio: para ello se deberá ejecutar periódicamente el comando `VACUUM`, que recuperará el espacio de disco de filas borradas o modificadas, actualizará las estadísticas utilizadas por el planificador de PostgreSQL y mejorará las condiciones de acceso.

2) Reindexar: en ciertos casos, PostgreSQL puede dar algunos problemas con la reutilización de los índices, por ello, es conveniente utilizar `REINDEX` periódicamente para eliminar páginas y filas. También se puede utilizar contrib/reindexdb para reindexar una DB entera (se debe tener en cuenta que dependiendo del tamaño de las DB, estos comandos pueden tardar un cierto tiempo).

3) Cambio de archivos de registro (*logs*): se debe evitar que los archivos de registro sean muy grandes y difíciles de manejar. Se puede hacer fácilmente cuando se inicia el servidor con `pg_ctl start | logrotate`. `logrotate` renombra y abre un nuevo archivo de registro y se puede configurar con `/etc/logrotate.conf`.

4) Copia de seguridad y recuperación (*backup* y *recovery*): existen dos formas de guardar los datos, con la sentencia SQL `dump` o guardando el archivo de la DB. El primero es `pg_dump ArchivoDB >ArchivoBackup`. Para recuperar, se puede utilizar `psql ArchivoDB <ArchivoBackup`. Para guardar todas las DB del servidor, se puede ejecutar `pg_dumpall >ArchivoBackupTotal`. Otra es-

trategia es guardar los archivos de las bases de datos a nivel del sistema operativo, por ejemplo con `tar -cf backup.tar /usr/local/pgsql/data`. Existen dos restricciones que pueden hacer que este método sea poco práctico:

a) el servidor debe detenerse antes de guardar y de recuperar los datos y

b) deben conocerse muy bien todas las implicaciones a nivel archivo, donde están todas las tablas, transacciones y demás, ya que de lo contrario, una DB puede quedar inútil. Además, por lo general, el tamaño que se guardará será mayor que el realizado con los métodos anteriores, ya que por ejemplo, con el `pg_dumb` no se guardan los índices, sino el comando para recrearlos.

1.1.6. Resumen de la instalación de PostgreSQL

```
./configure
gmake
su
gmake install
adduser postgres
mkdir /usr/local/pgsql/data
chown postgres /usr/local/pgsql/data
su - postgres
/usr/local/pgsql/bin/initdb -D /usr/local/pgsql/data
/usr/local/pgsql/bin/postgres -D /usr/local/pgsql/data >logfile 2>&1 &
/usr/local/pgsql/bin/createdb test
/usr/local/pgsql/bin/psql test
```

1.2. El lenguaje SQL

No es la finalidad de este subapartado hacer un tutorial sobre SQL, pero se analizarán unos ejemplos para ver las capacidades de este lenguaje. Son ejemplos que vienen con la propia distribución de PostgreSQL en el directorio `DirectorioInstalacion/src/tutorial`. Para acceder a ellos, cambiad al directorio de PostgreSQL (`cd DirectorioInstalación/src/tutorial`) y ejecutad `psql -s nteumdb` y después, dentro `ıbasics.sql`. El parámetro `ı` lee los comandos del archivo especificado (`basic.sql` en nuestro caso). PostgreSQL es una base de datos relacional (Relational Database Management System, RDBMS), lo cual significa que maneja los datos almacenados en tablas. Cada tabla tiene un número determinado de filas y de columnas, y cada columna tiene un tipo específico de datos. La tablas se agrupan en una DB y un único servidor maneja esta colección de DB (todo el conjunto se denomina agrupación o clúster de bases de datos, *database cluster*). Para crear, por ejemplo, una tabla con psql, ejecutad:

```
CREATE TABLE tiempo (
    ciudad varchar(80),
    temp_min int,
    temp_max int,
    lluvia real,
    dia date
);
```

El comando termina cuando se pone ';' y se pueden utilizar espacios en blanco y tabulaciones libremente. `Varchar(80)` especifica una estructura de datos que puede almacenar hasta 80 caracteres (en nuestro caso). El *point* es un tipo específico de PostgreSQL.

Para borrar la tabla:
`DROP TABLE nombre_tabla;`
Para introducir datos, se pueden utilizar dos formas: poner todos los datos de la tabla o indicar las variables y los valores que se desean modificar:
`INSERT INTO tiempo VALUES ('Barcelona', 16, 37, 0.25, '2007-03-19');`
`INSERT INTO tiempo (ciudad, temp_min, temp_max, lluvia, dia) VALUES ('Barcelona', 16, 37, 0.25, '2007-03-19');`
Esta forma puede ser sencilla para unos pocos datos, pero cuando hay que introducir gran cantidad de datos, se pueden copiar desde un archivo con la sentencia:
`COPY tiempo FROM '/home/user/tiempo.txt';` Este archivo debe estar en el servidor, no en el cliente). Para mirar una tabla, podríamos hacer:
`SELECT * FROM tiempo;`
donde el * significa "todas las columnas".
Ejemplo: introducir datos en tabla. Dentro de psql:
`INSERT INTO NombreTB (valorVar1, ValorVar2,...);`
Datos desde un archivo. Dentro de psql:
`COPY NombreTB FROM 'NombreArchivo';`
Visualizar datos. Dentro de psql:
`SELECT * FROM NombreTB;`
Algunos ejemplos de comandos más complejos serían (dentro de psql). Visualiza la columna ciudad después de realizar la operación:
`SELECT ciudad, (temp_max+temp_min)/2 AS temp_media, date FROM tiempo;`
Visualiza todo donde se cumple la operación lógica:
`SELECT * FROM tiempo WHERE city = 'Barcelona' AND lluvia > 0.0;`
Unión de tablas:
`SELECT * FROM tiempo, ciudad WHERE ciudad = nombre;`
Funciones, máximo en este caso:
`SELECT max(temp_min) FROM tiempo;`
Funciones anidadas:
`SELECT ciudad FROM tiempo WHERE temp_min = (SELECT max(temp_min) FROM tiempo);`
Modificación selectiva:
`UPDATE tiempo SET temp_max = temp_max 2, temp_min = temp_min 2 WHERE dia >'19990128';`
Borrado del registro:
`DELETE FROM tiempo WHERE ciudad = 'Sabadell';`
Otros comandos de utilidad:
Login como usuario postgres (SuperUser) para trabajar con la base de datos: `# su - postgres`
Crear bases de datos: `$ createdb nombre_BD`
Eliminar base de datos: `$ dropdb nombre_BD`
Acceder a la base de datos: `$ psql nombre_BD`
Ayuda (dentro de la base de datos; por ejemplo, mynteum: `mynteum=# \h`
Salir del psql: `mynteum=# \q`
Guardar la base de datos: `$ pg_dump mynteum >db.out` Cargar la base de datos guardada anteriormente: `$ psql -d mynteum -f db.out`
Para guardar todas las bases de datos: `# su - postgres` y después `$ pg_dumpall >/var/lib/pgsql/backups/dumpall.sql`
Para restaurar una base de datos: `# su - postgres` y después `$ psql -f /var/lib/pgsql/backups/dumpall.sql mynteum`
Para mostrar las bases de datos: `psql -l` o si no, desde dentro `mynteum=# \l;`
Mostrar los usuarios: desde dentro del comando `psql` ejecutar `mymydb=# SELECT * FROM "pg_user";`
Mostrar las tablas: desde dentro del comando `psql` ejecutar `mynteum=# SELECT * FROM "pg_tables";`
Cambiar la contraseña: desde dentro del comando `psql` `mynteum=# UPDATE pg_shadow SET passwd = 'new_password' where usename = 'username';` o también `ALTER USER nombre WITH PASSWORD 'password';`
Limpiar todas las bases de datos: `$ vacuumdb - -quiet - -all`

1.2.1. Entornos de administración gráficos

Existen diversos entornos de administración gráficos, como **Phppgadmin*** y **Pgadmin****. Otra aplicación que funciona pero que ha dejado de ser mantenida (y por lo cual no es recomendable) es **pgaccess*****. Estos programas permiten acceder y administrar una base de datos con una interfaz gráfica. Para la mayoría, la forma más fácil de acceder es desde una terminal. El administrador de la DB (si no es el usuario postgresql) deberá hacer `xhost +`, lo cual permite que otras aplicaciones puedan conectarse al *display* del usuario actual y, a continuación, ejecutar el nombre de la aplicación. Entre todos estos entornos, **Phppgadmin** presenta muy buenas características (está incluido en la mayoría de distribuciones) y una vez instalado, se accede a él mediante `http://localhost/phppgadmin`. Después de configurar el lenguaje, podremos seleccionar el servidor, el usuario y la contraseña para la base de datos. Es recomendable crear un usuario con contraseña para acceder a la base de datos desde el usuario postgres del sistema operativo (por ejemplo, `su - postgres`) con el comando `psql` y hacer `CREATE USER admin WITH PASSWD 'poner_passwd';`. Para ver que todo es correcto, se puede hacer `SELECT * FROM "pg_user";` y se verá el usuario creado con * en la contraseña. En Phppgadmin podremos seleccionar a la izquierda el servidor y configurar los parámetros indicando que el servidor es *localhost* y el usuario y la contraseña, los creados anteriormente. A partir de aquí, veremos las bases de datos, así como todos los parámetros y configuraciones.

***http://phppgadmin.sourceforge.net**
****http://www.pgadmin.org**
*****http://sourceforge.net/projects/pgaccess**

Otra herramienta interesante es **Webmin**. Es una interfaz basada en la web para administrar sistemas para Unix que permite configurar usuarios, Apache, DNS, compartir archivos, servidores de bases de datos, etc. La ventaja de Webmin, sobre todo para usuarios que se inician en la administración, es que elimina la necesidad de editar manualmente los archivos de configuración y permite administrar un sistema de forma local o remota.

Enlaces de interés

Webmin no está disponible en algunas distribuciones y se debe bajar e instalar directamente de la web: http://www.webmin.com. Toda la documentación y la información de los módulos de Webmin disponibles se encuentra en: http://doxfer.webmin.com/Webmin.

1.3. MySQL

MySQL [7] es, según sus autores, la base de datos (DB) SQL abierta, es decir, de software libre (*Open Source*) más popular. En la actualidad, es propiedad de Oracle, que es quien realiza las inversiones y mantenimientos y que a partir de la versión (aún en fase Beta) de MySQL 5.5 (primera versión después de la compra de la compañía Sun), ha insistido en que continuará invirtiendo en esta base de datos. En esta versión, se ha integrado el motor de persistencia InnoDB y se han añadido mejoras internas para llegar a incrementar el rendimiento hasta en un 200 %, además de lograr tiempos de recuperación 10 veces menores que en la versión anterior. MySQL es un DBMS (*Database Management System*). Un DBMS es el que puede añadir y procesar los datos almacenados dentro de la DB.

Enlace de interés

Toda la documentación sobre MySQL se puede obtener desde la página web siguiente: http://dev.mysql.com/usingmysql/get_started.html.

Al igual que PostgreSQL, MySQL es una base de datos relacional, es decir, que almacena los datos en tablas en lugar de en una única ubicación, lo cual permite aumentar la velocidad y la flexibilidad.

Al ser software libre, cualquiera puede obtener el código, estudiarlo y modificarlo de acuerdo con sus necesidades sin pago alguno, ya que MySQL utiliza licencia GPL. MySQL provee en su página web un conjunto de estadísticas y prestaciones en comparación con otras DB para mostrar al usuario cuán rápida, fiable y fácil es de usar. La decisión de elegir una DB se debe hacer cuidadosamente en función de las necesidades de los usuarios y del entorno donde se utilizará esta DB.

1.3.1. Instalación de MySQL

MySQL se puede obtener desde la página web* del proyecto o desde cualquiera de los repositorios de software. Se pueden obtener los binarios y los archivos fuente para compilarlos e instalarlos. En el caso de los binarios, utilizar la distribución de Debian y seleccionar los paquetes `mysql-*` (`client`, `server` y `common` son necesarios). La instalación, después de unas preguntas, creará un usuario mysql y una entrada en `/etc/init.d/mysql` para arrancar o detener el servidor en el boot. También se puede hacer manualmente:

*http://www.mysql.com

/etc/init.d/mysql start|stop

Para acceder a la base de datos, se puede utilizar el monitor mysql desde la línea de comandos. Si obtiene los binarios (no Debian ni RPM, con esto simplemente utilizad las utilidades comunes apt-get y rpm), por ejemplo un archivo comprimido desde el sitio web de MySQL, deberá ejecutar los siguientes comandos para instalar la DB:

```
groupadd mysql
useradd -g mysql mysql
cd /usr/local
gunzip </path/to/mysql-VERSION-OS.tar.gz | tar xvf -
ln -s full-path-to-mysql-VERSION-OS mysql
cd mysql
scripts/mysql_install_db –user=mysql
chown -R root .
chown -R mysql data
chgrp -R mysql .
bin/mysqld_safe –user=mysql &
```

Esto crea el usuario/grupo/directorio, descomprime e instala la DB en el directorio `/usr/local/mysql`. En caso de obtener el código fuente, los pasos son similares:

```
groupadd mysql
useradd -g mysql mysql
gunzip <mysql-VERSION.tar.gz | tar -xvf -
cd mysql-VERSION
./configure - -prefix=/usr/local/mysql
make
make install
cp support-files/my-medium.cnf /etc/my.cnf
cd /usr/local/mysql
bin/mysql_install_db –user=mysql
chown -R root .
chown -R mysql var
chgrp -R mysql .
bin/mysqld_safe –user=mysql &
```

Es importante prestar atención cuando se realiza la configuración, ya que `prefix=/usr/local/mysql` es el directorio donde se instalará la DB y se puede cambiar para ubicar la DB en el directorio que se desee.

1.3.2. Postinstalación y verificación de MySQL

Una vez realizada la instalación (ya sea de los binarios o del código fuente), se deberá verificar si el servidor funciona correctamente. En Debian se puede hacer directamente (el usuario es el que se ha definido durante la instalación a igual que su contraseña, indicados en los comandos por `-u` y `-p`, respectivamente):

```
/etc/init.d/mysql start      Inicia el servidor
mysqladmin -u user -p version      Genera información de versiones
mysqladmin -u user -p variables      Muestra los valores de las variables
mysqladmin -u root -p shutdown      Finaliza la ejecución del servidor
mysqlshow -u user -p      Muestra las DB predefinidas
mysqlshow mysql -u user -p      Muestra las tablas de la DB mysql
```

Si se instala desde el código fuente, antes de hacer estas comprobaciones se deben ejecutar los siguientes comandos para crear las bases de datos (desde el directorio de la distribución):

```
./scripts/mysql_install_db
cd DirectorioInstalacionMysql
./bin/mysqld_safe –user = mysql &
```

Si se instala desde binarios (RPM, Pkg ...), se debe hacer lo siguiente:

```
cd DirectorioInstalacionMysql
./scripts/mysql_install_db
./bin/mysqld_safe user = mysql &
```

El *script* `mysql_install_db` crea la DB `mysql` y `mysqld_safe` arranca el servidor `mysqld`. A continuación, se pueden probar todos los comandos dados anteriormente para Debian, excepto el primero, que es el que arranca el servidor. Además, si se han instalado los *tests*, se podrán ejecutar con `cd sql-bench` y después, con `run-all-tests`. Los resultados se encontrarán en el directorio `sql-bech/Results` para compararlos con otras DB.

1.3.3. El programa monitor (cliente) mysql

El cliente `mysql` se puede utilizar para crear y utilizar DB simples, es interactivo y permite conectarse al servidor, ejecutar búsquedas y visualizar los resultados. También funciona en modo *batch* (como un *script*) donde los comandos se le pasan a través de un archivo. Para ver todas las opciones del comando, se puede ejecutar `mysql -help`. Podremos realizar una conexión (local o remota) con el comando mysql, por ejemplo, para una conexión por la interfaz de red, pero desde la misma máquina:

```
mysql -h localhost -u usuario -p [NombreDB]
```

Si no se pone el último parámetro, no se selecciona ninguna DB. Una vez dentro, el `mysql` pondrá un *prompt* (**mysql**>) y esperará a que le introduzcamos algún comando (propio y SQL), por ejemplo, *help*. A continuación, daremos una serie de comandos para probar el servidor (recordad poner siempre el ';' para terminar el comando):

```
  mysql>SELECT VERSION(), CURRENT_DATE;
```
Se pueden utilizar mayúsculas o minúsculas.
```
  mysql>SELECT SIN(PI()/4), (4+1)*5;
```
Calculadora.
```
  mysql>SELECT VERSION(); SELECT NOW();
```
Múltiples comandos en la misma línea
```
  mysql>SELECT
   ->USER()
   ->,
   ->CURRENT_DATE;
```
o en múltiples líneas.
```
  mysql>SHOW DATABASES;
```
Muestra las DB disponibles.
```
  mysql>USE test
```
Cambia la DB.
```
  mysql>CREATE DATABASE nteum; USE nteum;
```
Crea y selecciona una DB llamada nteum.
```
  mysql>CREATE TABLE pet (name VARCHAR(20), owner VARCHAR(20),
  ->species VARCHAR(20), sex CHAR(1), birth DATE, death DATE);
```
Crea una tabla dentro de nteum.
```
  mysql>SHOW TABLES;
```
Muestra las tablas.
```
  mysql>DESCRIBE pet;
```
Muestra la definición de la tabla.
```
  mysql>LOAD DATA LOCAL INFILE ”pet.txt” INTO TABLE pet;
```
Carga datos desde `pet.txt` en pet. El archivo `pet.txt` debe tener un registro por línea separado por tabulaciones de los datos, de acuerdo con la definición de la tabla (fecha en formato AAAA-MM-DD)
```
  mysql>INSERT INTO pet
  ->VALUES ('Marciano','Estela','gato','f','1999-03-30',NULL);
```
Carga los datos *in-line*.
```
  mysql>SELECT * FROM pet;
```
Muestra los datos de la tabla.
```
  mysql>UPDATE pet SET birth = “1989-08-31” WHERE name = “Browser”;
```
Modifica los datos de la tabla.
```
  mysql>SELECT * FROM pet WHERE name = “Browser”;
```
Muestra selectiva.
```
  mysql>SELECT name, birth FROM pet ORDER BY birth;
```
Muestra ordenada.
```
  mysql>SELECT name, birth FROM pet WHERE MONTH(birth) = 5;
```
Muestra selectiva con funciones.
```
  mysql>GRANT ALL PRIVILEGES ON *.* TO marciano@localhost ->IDENTIFIED
BY 'passwd' WITH GRANT OPTION;
```

Crea un usuario marciano en la DB. Lo debe hacer el root de la DB. También se puede hacer directamente con:

```
mysql>INSERT INTO user (Host,User,Password) ->
VALUES('localhost','marciano','passwd');
```

1.3.4. Caso de uso: procesamiento de datos con MySQL

Consideraremos los datos del Banco Mundial de datos y concretamente los de Internet: ancho de banda internacional (bits por persona). Desde la dirección http://datos.bancomundial.org/indicador podemos descargar una hoja de datos que tendrá una información como (la información muestra el parcial de un total de 196 países y desde el año 1960):

País ... 2000 2001 2002 2003 2004 2005 2006 2007 2008 2009 2010
...
España 297,14 623,61 1126,83 1938,19 2821,65 2775,71 6058,60 11008,05...

Se creará una base de datos en MySQL para importar estos datos y generar los listados que calculen la media por país y la media anual, y ordenen de mayor a menor el volumen para el año 2008; y un listado de los 10 países con mayor utilización de Internet por año. Sobre estos últimos datos, se debe hacer una clasificación de los cinco países con mayor utilización de Internet desde que existen datos (se muestran los comandos más relevantes y no para todos los datos).

```
mysql -u root -p
Welcome to the MySQL monitor. Commands end with ; or \g.
Server version: 5.0.51a24+ lenny3 (Debian)
Type 'help;' or '\h' for help. Type '\c' to clear the buffer.
mysql> create database bancomundial;

mysql> show databases;
+--------------------+
| Database           |
+--------------------+
| information_schema |
| bancomundial       |
| mysql              |
+--------------------+
3 rows in set (0.00 sec)

mysql> use bancomundial;
Database changed
mysql> create table paisbai(pais varchar(100), 1960
varchar(100), 1961 varchar(100), 1962 varchar(100), 1963
varchar(100), 1964 varchar(100), 1965 varchar(100), 1966
varchar(100), 1967 varchar(100), 1968 varchar(100), 1969
varchar(100), 1970 varchar(100), 1971 varchar(100), 1972
varchar(100), 1973 varchar(100), 1974 varchar(100), 1975
varchar(100), 1976 varchar(100), 1977 varchar(100), 1978
varchar(100), 1979 varchar(100), 1980 varchar(100), 1981
varchar(100), 1982 varchar(100), 1983 varchar(100), 1984
varchar(100), 1985 varchar(100), 1986 varchar(100), 1987
varchar(100), 1988 varchar(100), 1989 varchar(100), 1990
varchar(100), 1991 varchar(100), 1992 varchar(100), 1993
varchar(100), 1994 varchar(100), 1995 varchar(100), 1996
varchar(100), 1997 varchar(100), 1998 varchar(100), 1999
varchar(100), 2000 varchar(100), 2001 varchar(100), 2002
```

```
varchar(100), 2003 varchar(100), 2004 varchar(100), 2005
varchar(100), 2006 varchar(100), 2007 varchar(100), 2008
varchar(100), 2009 varchar(100), 2010 varchar(100) );

mysql>
CREATE TABLE ‘bancomundial‘.‘paisbai‘ (‘pais‘ VARCHAR(100) NOT
NULL, ‘1960‘ VARCHAR(100) NOT NULL, ‘1961‘ VARCHAR(100) NOT
NULL, ‘1962‘ VARCHAR(100) NOT NULL, ‘1963‘ VARCHAR(100) NOT
NULL, ‘1964‘ VARCHAR(100) NOT NULL, ‘1965‘ VARCHAR(100) NOT
NULL, ‘1966‘ VARCHAR(100) NOT NULL, ‘1967‘ VARCHAR(100) NOT
NULL, ‘1968‘ VARCHAR(100) NOT NULL, ‘1969‘ VARCHAR(100) NOT
NULL, ‘1970‘ VARCHAR(100) NOT NULL, ‘1971‘ VARCHAR(100) NOT
NULL, ‘1972‘ VARCHAR(100) NOT NULL, ‘1973‘ VARCHAR(100) NOT
NULL, ‘1974‘ VARCHAR(100) NOT NULL, ‘1975‘ VARCHAR(100) NOT
NULL, ‘1976‘ VARCHAR(100) NOT NULL, ‘1977‘ VARCHAR(100) NOT
NULL, ‘1978‘ VARCHAR(100) NOT NULL, ‘1979‘ VARCHAR(100) NOT
NULL, ‘1980‘ VARCHAR(100) NOT NULL, ‘1981‘ VARCHAR(100) NOT
NULL, ‘1982‘ VARCHAR(100) NOT NULL, ‘1983‘ VARCHAR(100) NOT
NULL, ‘1984‘ VARCHAR(100) NOT NULL, ‘1985‘ VARCHAR(100) NOT
NULL, ‘1986‘ VARCHAR(100) NOT NULL, ‘1987‘ VARCHAR(100) NOT
NULL, ‘1988‘ VARCHAR(100) NOT NULL, ‘1989‘ VARCHAR(100) NOT
NULL, ‘1990‘ VARCHAR(100) NOT NULL, ‘[...]
mysql> describe paisbai;
+-------------------------------------+
|Field | Type         |Null |K| Def. |E|
| pais | varchar(100) | YES | | NULL | |
| 1960 | varchar(100) | YES | | NULL | |
| 1961 | varchar(100) | YES | | NULL | |
| 1962 | varchar(100) | YES | | NULL | |
| 1963 | varchar(100) | YES | | NULL | |
| 1964 | varchar(100) | YES | | NULL | |
| 1965 | varchar(100) | YES | | NULL | |
...
| 2009 | varchar(100) | YES | | NULL | |
| 2010 | varchar(100) | YES | | NULL | |
+-------------------------------------+

1. Para cargar los datos, se puede entrar en http://localhost/phpmyadmin e
importar los datos desde un fichero a la base de datos bancomundial o desde la línea del mysql.
LOAD DATA LOCAL INFILE ’/tmp/phpO5Dhpl’ REPLACE INTO TABLE ‘paisbai‘
FIELDS TERMINATED BY ’;’

ENCLOSED BY ’"’

ESCAPED BY ’\\’
LINES TERMINATED BY ’\n’;

2. Listado que permite calcular la media por país.
consulta SQL: SELECT ‘pais‘,
(‘1960‘+‘1961‘+‘1962‘+‘1963‘+‘1964‘+‘1965‘+‘1966‘+‘1967‘+‘1968‘+
‘1969‘+‘1970‘+‘1971‘+‘1972‘+‘1973‘+‘1974‘+‘1975‘+‘1976‘+‘1977‘+‘
1978‘+‘1979‘+‘1980‘+‘1981‘+‘1982‘+‘1983‘+‘1984‘+‘1985‘+‘1986‘+‘1
987‘+‘1988‘+‘1989‘+‘1990‘+‘1991‘+‘1992‘+‘1993‘+‘1994‘+‘1995‘+‘19
96‘+‘1997‘+‘1998‘+‘1999‘+‘2000‘+‘2001‘+‘2002‘+‘2003‘+‘2004‘+‘200
5‘+‘2006‘+‘2007‘+‘2008‘+‘2009‘+‘2010‘)/51 as mediapais FROM
‘paisbai‘ LIMIT 0, 30 ;
Filas: 30
pais        mediapais
Afganistán  0.0203921568627
Albania     7.3696078431373
Argelia     0.4425490196078

3. Generar un listado que visualice la media anual.
consulta SQL: SELECT AVG (‘1989‘) AS ‘1989‘, AVG(‘1990‘) as
‘1990‘, AVG(‘1991‘) as ‘1991‘, AVG(‘1992‘) as ‘1992‘,
AVG(‘1993‘) as ‘1993‘, AVG(‘1994‘) as ‘1994‘, AVG(‘1995‘) as
‘1995‘, AVG(‘1996‘) as ‘1996‘, AVG(‘1997‘) as ‘1997‘,
AVG(‘1998‘) as ‘1998‘, AVG(‘1999‘) as ‘1999‘, AVG(‘2000‘) as
‘2000‘, AVG(‘2001‘) as ‘2001‘, AVG(‘2002‘) as ‘2002‘,
AVG(‘2003‘) as ‘2003‘, AVG(‘2004‘) as ‘2004‘, AVG(‘2005‘) as
‘2005‘, AVG(‘2006‘) as ‘2006‘, AVG(‘2007‘) as ‘2007‘,AVG(‘2008‘)
as ‘2008‘, AVG(‘2009‘) as ‘2009‘, AVG(‘2010‘) as ‘2010‘ FROM
‘paisbai‘;
```

```
Filas: 1
1989 1990 1991 1992 1993 1994 1995 1996 1997 1998 1999 2000
0.0001 0.000 0.000 0.001 0.0019 0.005 0.044 0.158 0.6547 1.3870 27.483 126.9760 387.70
```

3. Generar un listado que visualice el orden de mayor a menor volumen para el año 2008.

```
SELECT ‘pais‘, ‘2008‘ FROM ‘paisbai‘ ORDER BY ‘2008‘ DESC LIMIT 0 , 30;
```

```
+--------------------+
| pais     | 2008    |
| Lituania | 9751.01 |
| Mongolia | 946.53  |
| Rumania  | 9110.51 |
| Zimbabwe | 9.71    |
...
| Bhután   | 65.52   |
| Hungría  | 5977.17 |
| Viet Nam | 580.72  |
+--------------------+
```

4. Generar un listado de los 10 países con mayor utilización de Internet por año.

Alternativa 1.

```
SELECT ‘pais‘ , SUM( ‘1989‘ ) AS ‘1989‘ , SUM( ‘1990‘ ) AS
‘1990‘ , SUM( ‘1991‘ ) AS ‘1991‘ , SUM( ‘1992‘ ) AS ‘1992‘ ,
SUM( ‘1993‘ ) AS ‘1993‘ , SUM( ‘1994‘ ) AS ‘1994‘ ,
SUM( ‘1995‘ ) AS ‘1995‘ , SUM( ‘1996‘ ) AS ‘1996‘ ,
SUM( ‘1997‘ ) AS ‘1997‘ , SUM( ‘1998‘ ) AS ‘1998‘ ,
SUM( ‘1999‘ ) AS ‘1999‘ , SUM( ‘2000‘ ) AS ‘2000‘ ,
SUM( ‘2001‘ ) AS ‘2001‘ , SUM( ‘2002‘ ) AS ‘2002‘ ,
SUM( ‘2003‘ ) AS ‘2003‘ , SUM( ‘2004‘ ) AS ‘2004‘ ,
SUM( ‘2005‘ ) AS ‘2005‘ , SUM( ‘2006‘ ) AS ‘2006‘ ,
SUM( ‘2007‘ ) AS ‘2007‘ , SUM( ‘2008‘ ) AS ‘2008‘ ,
SUM( ‘2009‘ ) AS ‘2009‘ , SUM( ‘2010‘ ) AS ‘2010‘
FROM ‘paisbai‘
ORDER BY ‘pais‘ DESC
LIMIT 0 , 10;
```

Alternativa 2:

```
SELECT ‘pais‘ , MAX( ‘1989‘ ) AS ‘1989‘ , MAX( ‘1990‘ ) AS
‘1990‘ , MAX( ‘1991‘ ) AS ‘1991‘ , MAX( ‘1992‘ ) AS ‘1992‘ ,
MAX( ‘1993‘ ) AS ‘1993‘ , MAX( ‘1994‘ ) AS ‘1994‘ ,
MAX( ‘1995‘ ) AS ‘1995‘ , MAX( ‘1996‘ ) AS ‘1996‘ ,
MAX( ‘1997‘ ) AS ‘1997‘ , MAX( ‘1998‘ ) AS ‘1998‘ ,
MAX( ‘1999‘ ) AS ‘1999‘ , MAX( ‘2000‘ ) AS ‘2000‘ ,
MAX( ‘2001‘ ) AS ‘2001‘ , MAX( ‘2002‘ ) AS ‘2002‘ ,
MAX( ‘2003‘ ) AS ‘2003‘ , MAX( ‘2004‘ ) AS ‘2004‘ ,
MAX( ‘2005‘ ) AS ‘2005‘ , MAX( ‘2006‘ ) AS ‘2006‘ ,
MAX( ‘2007‘ ) AS ‘2007‘ , MAX( ‘2008‘ ) AS ‘2008‘ ,
MAX( ‘2009‘ ) AS ‘2009‘ , MAX( ‘2010‘ ) AS ‘2010‘
FROM ‘paisbai‘
GROUP BY ‘pais‘
LIMIT 0 , 10;
```

```
+----------------+------------------+
| país           | promedio         |
+----------------+------------------+
| Luxemburgo     |    284474.679628 |
| Hong Kong      | 21468.6420499333 |
| San Marino     | 5464.22342423529 |
| Países Bajos   | 3949.18559792941 |
| Dinamarca      |     3743.7899214 |
| Estonia        | 3118.98744675686 |
| Suecia         | 2967.78367829608 |
| Reino Unido    | 1902.25120777059 |
| Suiza          | 1897.13803142745 |
| Bélgica        | 1781.95881669216 |
+----------------+------------------+
```

1.3.5. Administración de MySQL

MySQL dispone de un archivo de configuración en `/etc/mysql/my.cnf` (en Debian), al que se pueden cambiar las opciones por defecto de la DB, como por ejemplo, el puerto de conexión, el usuario, la contraseña de los usuarios remotos, los archivos de registro (*logs*), los archivos de datos, si acepta conexiones externas, etc. Con respecto a la seguridad, se deben tomar algunas precauciones:

1) No dar a nadie (excepto al usuario root de Mysql) acceso a la tabla `user` dentro de la DB mysql, ya que aquí se encuentran las contraseñas de los usuarios que podrían utilizarse con otros fines.

2) Verificar `mysql -u root`. Si se puede acceder, significa que el usuario root no tiene contraseña. Para cambiarlo, se puede hacer:

```
mysql -u root mysql
mysql>UPDATE user SET Password = PASSWORD('new_password')
->WHERE user = 'root';
mysql>FLUSH PRIVILEGES;
```

Ahora, para conectarse como root: `mysql -u root -p mysql`.

3) Comprobar la documentación* respecto a las condiciones de seguridad y del entorno de red para evitar problemas de ataques o intrusiones.

***http://dev.mysql.com /usingmysql/get_started.html**

4) Para hacer copias de la base de datos, se puede utilizar el comando:

```
mysqldump - -tab = /DirectorioDestino - -opt NombreDB
```

o también:

```
mysqlhotcopy NombreDB /DirectorioDestino
```

Asimismo, se pueden copiar los archivos con extensión FRM, MYD, y MYI con el servidor parado. Para recuperar, se puede ejecutar mediante el comando `REPAIR TABLE` o `myisamchk -r`, lo cual funcionará en la mayoría de las veces. En caso contrario, se podrían copiar los archivos guardados y arrancar el servidor. Existen otros métodos alternativos en función de lo que se quiera recuperar, como la posibilidad de guardar/recuperar parte de la DB (consultad la documentación).

1.3.6. Interfaces gráficas

Para Mysql hay gran cantidad de interfaces gráficas, incluso propias del paquete MySQL. Una de ellas es **Mysql Administrator**, que es una aplicación

potente para la administración y control de bases de datos basadas en MySQL. Esta aplicación integra la gestión, el control y el mantenimiento de forma simple y en un mismo entorno de la BD. Las características principales son: administración avanzada de grandes DB, reducción de errores a través de una "administración visual", mayor productividad y un entorno seguro de gestión. En el sitio original*, estas aplicaciones ahora forman parte del paquete MySQL Workbench, que provee herramientas para modelar y diseñar la base de datos, desarrollar SQL (equivalente al MySQL Browser) y administrar la base de datos (equivalente a MySQL Administrator).

Enlace de interés

Se puede encontrar toda la documentación para la instalación y puesta en marcha de Mysql Administrator en http://dev.mysql.com /doc/administrator/en /index.html.

***http://www.mysql.com /downloads/workbench**

Como herramienta de administración, también recomendamos **phpMyAdmin** (incluida en la mayoría de las distribuciones), que es una herramienta de software libre escrito en PHP para gestionar la administración de MySQL a través de la WWW. phpMyAdmin es compatible con un amplio conjunto de operaciones en MySQL, como por ejemplo, gestión de bases de datos, tablas, campos, relaciones, índices, usuarios, permisos, etc. y también tiene la capacidad de ejecutar directamente cualquier sentencia SQL. Su instalación se puede hacer desde la gestión de programas de distribución de trabajo (apt/rpm, yum/Synaptic, etc.) que durante el proceso de instalación pedirá con qué servidores de WWW se quiere hacer la integración. Una vez instalada, se puede iniciar a través de `http://localhost/phpmyadmin`, que pedirá el usuario y la contraseña del servidor (indicados en los pasos anteriores).

Existen otras herramientas que permiten hacer tareas similares, como **Webmin**, que permite gestionar y administrar bases de datos MySQL (incluyendo el módulo correspondiente). Si bien este paquete ya no se incluye con algunas distribuciones, se puede descargar desde http://www.webmin.com. Durante la instalación, el Webmin avisará de que el usuario principal será el root y utilizará la misma contraseña que el root del sistema operativo. Será posible conectarse, por ejemplo, desde un navegador `https://localhost:10000`, el cual solicitará aceptar (o denegar) la utilización del certificado para la comunicación SSL y, a continuación, mostrará todos los servicios que puede administrar, entre ellos Mysql Data Base Server.

1.4. Source Code Control System (CVS y Subversion)

El **Concurrent Versions System** (CVS) es un sistema de control de versiones que permite mantener versiones antiguas de archivos (generalmente código fuente), guardando un registro (*log*) de quién, cuándo y porqué fueron realizados los cambios.

A diferencia de otros sistemas, CVS no trabaja con un archivo/directorio por vez, sino que actúa sobre colecciones jerárquicas de los directorios que contro-

la. El CVS tiene por objetivo ayudar a gestionar versiones de software y controla la edición concurrente de archivos fuente por múltiples autores. El CVS utiliza internamente otro paquete llamado **RCS** (Revision Control System) como una capa de bajo nivel. Si bien el RCS puede ser utilizado independientemente, esto no se aconseja, ya que CVS, además de su propia funcionalidad, presenta todas las prestaciones de RCS, pero con notables mejoras en cuanto a la estabilidad, el funcionamiento y el mantenimiento. Entre ellas, cabe destacar: funcionamiento descentralizado (cada usuario puede tener su propio árbol de código), edición concurrente, comportamiento adaptable mediante *shell scripts*, etc. [2, 22, 12].

Como ya se ha explicado en la introducción, **Subversion*** es un software de sistema de control de versiones diseñado específicamente para reemplazar al popular CVS y ampliar sus capacidades. Es software libre bajo una licencia de tipo Apache/BSD y se lo conoce también como **svn** por el nombre en línea de comandos.

*http://subversion.apache.org

Una característica importante de Subversion es que, a diferencia de CVS, los archivos versionados no tienen cada uno un número de revisión independiente sino que, todo el repositorio tiene un único número de versión que identifica un estado común de todos los archivos del repositorio en el tiempo que se ha "versionado".

Entre las características principales podemos mencionar:

1) se sigue la historia de los archivos y directorios a través de copias y renombrados;

2) las modificaciones atómicas y seguras (incluidos cambios a varios archivos);

3) la creación de ramas y etiquetas es eficiente y simple;

4) se envían solo las diferencias en ambas direcciones (en CVS siempre se envían al servidor archivos completos);

5) puede ser servido, mediante Apache, sobre WebDAV/DeltaV;

6) gestiona eficientemente archivos binarios (a diferencia de CVS, que los trata internamente como si fueran de texto).

Lectura recomendada

Existe un libro interesante (de libre distribución) que explica todo lo referente al Subversion. Está disponible en http://svnbook.red-bean.com/index.es.html y su traducción está bastante avanzada (http://svnbook.red-bean.com/nightly/es/index.html).

1.4.1. Revision Control System (RCS)

Como que CVS se basa en RCS y en algunos sistemas todavía se utiliza, se darán unas breves explicaciones. El RCS está formado por un conjunto de programas para las diferentes actividades del RCS: `rcs` (programa que controla los

atributos de los archivos bajo RCS), `ci` y `co` (verifican la entrada y la salida de los archivos bajo el control de RCS), `ident` (busca en el RCS los archivos por palabras claves/atributos), `rcsclean` (limpia archivos no utilizados o que no han cambiado), `rcsdiff` (ejecuta el comando `diff` para comparar versiones), `rcsmerge` (une dos ramas [archivos] en un único archivo) y `rlog` (imprime los mensajes de *log*). El formato de los archivos almacenados por RCS puede ser texto u otro formato, como por ejemplo binario. Un archivo RCS consiste en un archivo de revisión inicial llamado 1.1 y una serie de archivos de cambios, uno por cada revisión. Cada vez que se realiza una copia del repositorio hacia el directorio de trabajo con el comando `co` (obtiene una revisión de cada archivo RCS y lo pone en el archivo de trabajo) o se utiliza `ci` (almacena nuevas revisiones en el RCS), el número de versión se incrementa (por ejemplo, 1.2, 1.3...). Generalmente, los archivos están en el directorio `./RCS` y es necesario que el sistema operativo tenga instalados los comandos `diff` y `diff3` para que funcione adecuadamente. En Debian no es necesario compilarlo ya que se incluye en la distribución.

Con el comando `rcs` crearemos y modificaremos los atributos de los archivos (consultad `man rcs`). La forma más fácil de crear un repositorio es hacer en primer lugar un `mkdir rcs` en el directorio de originales e incluir los originales en el repositorio con: `ci nombre_archivos_fuentes`. Se puede utilizar el `*` y se recomienda tener siempre una copia de resguardo para evitar problemas. Esto creará las versiones de los archivos con nombre `./RCS/nombre_archivo` y solicitará un texto para describir el archivo. A continuación, con `co RCS/nombre_archivo`, obtendremos una copia de trabajo desde el repositorio. Se puede bloquear o desbloquear este archivo para evitar modificaciones con: `rcs -L nombre_archivo_de_trabajo` y `rcs -U nombre_archivo_de_trabajo`, respectivamente. Finalmente con `rlog nombre_del_archivo` podremos ver la información sobre las diferentes versiones [12].

1.4.2. Concurrent Versions System (CVS)

En primer lugar, se debe instalar el Concurrent Versions System (CVS) desde la distribución teniendo en cuenta que debemos tener instalado RCS y que deberemos instalar también OpenSSH, si se quiere utilizar conjuntamente con CVS para acceso remoto. Las variables de entorno `EDITOR CVSROOT` deben estar inicializadas, por ejemplo, en `/etc/profile` (o en `.bashrc` o `.profile`):

```
export EDITOR = /bin/vi
export CVSROOT = /usr/local/cvsroot
```

Obviamente, los usuarios pueden modificar estas definiciones utilizando /.bashrc y se debe crear el directorio donde estará el repositorio y configurar los permisos; como *root*, hay que hacer, por ejemplo:

```
export CVSROOT = /usr/local/cvsroot
groupadd cvs
useradd -g cvs -d $CVSROOT cvs
```

```
mkdir $CVSROOT
chgrp -R cvs $CVSROOT
chmod o-rwx $CVSROOT
chmod ug+rwx $CVSROOT
```

Para inicializar el repositorio y poner archivo de código en él:

```
cvs -d /usr/local/cvsroot init
```

`cvs init` tendrá en cuenta no sobrescribir nunca un repositorio ya creado para evitar pérdidas de otros repositorios. Luego, se deberán agregar los usuarios que trabajarán con el CVS al grupo cvs; por ejemplo, para agregar el usuario nteum:

```
usermod -G cvs,nteum
```

Ahora el usuario nteum deberá introducir sus archivos en el directorio del repositorio (`/usr/local/cvsroot` en nuestro caso):

```
export EDITOR = /bin/vi
export CVSROOT = /usr/local/cvsroot
export CVSREAD = yes
cd directorio_de_originales
cvs import NombreDelRepositorio vendor_1_0 rev_1_0
```

El nombre del repositorio puede ser un identificador único o también puede ser `usuario/proyecto/xxxx` si es que el usuario desea tener organizados sus repositorios. Esto creará un árbol de directorios en `CVSROOT` con esa estructura y añade un directorio (`/usr/local/cvsroot/NombreDelRepositorio`) en el repositorio con los archivos que a partir de este momento estarán en el repositorio. Una prueba para saber si se ha almacenado todo correctamente es almacenar una copia en el repositorio, crear después una copia desde allí y comprobar las diferencias. Por ejemplo, si los originales están en el `directorio_del_usuario/dir_org` y se desea crear un repositorio como `primer_cvs/proj`, se deberán ejecutar los siguientes comandos:

`cd dir_org` *Cambiar al directorio del código fuente original.*

`cvs import -m "Fuentes originales" primer_cvs/proj usuarioX vers0`

Crea el repositorio en `primer_cvs/proj` con `usuarioX` y `vers0`.

`cd..` *Cambiar al directorio superior de dir_org.*

`cvs checkout primer_cvs/proj`

Generar una copia del repositorio. La variable `CVSROOT` debe estar inicializada, de lo contrario, se deberá indicar todo el camino.

`diff -r dir_org primer_cvs/proj`

Muestra las diferencias entre uno y otro. No deberá haber ninguna excepto por el directorio `primer_cvs/proj/CVS` que ha creado el CVS.

`rm -r dir_org`

Borra los originales (realizad siempre una copia de resguardo por motivos de seguridad y para tener una referencia de dónde se inició el trabajo con el CVS).

```
Estructura de los directorios, archivos y ramas:
/home/nteum/primer_cvs/proj: a.c b.c c.c Makefile
            usuarioX, vers0.x  -> Trabajar solo con este directorio después del \emph{import}
/home/nteum/dir_org/: a.c b.c c.c Makefile              Después del \emph{checkout}, deberá borrarse

/usr/local/cvsroot/primer_cvs/proj/: a.c b.c c.c Makefile usuarioX, vers0.1
/usr/local/cvsroot/primer_cvs/proj/: a.c b.c c.c Makefile usuarioX, vers0.1.1  -> Branch 1.1

/usr/local/cvsroot/primer_cvs/proj/: a.c b.c c.c Makefile usuarioX, vers0.2
/usr/local/cvsroot/primer_cvs/proj/: a.c b.c c.c Makefile usuarioX, vers0.x
```

El hecho de borrar los originales no siempre es una buena idea, salvo en este caso, después de que se haya verificado que están en el repositorio, para que no se trabaje sobre ellos por descuido y para que los cambios no queden reflejados sobre el CVS. Sobre máquinas donde los usuarios quieren acceder (por `ssh`) a un servidor CVS remoto, se deberá hacer:

```
export CVSROOT = ":ext:user@CVS.server.com:/home/cvsroot";
export CVS_RSH = "ssh",
```

donde `user` es el *login* del usuario y `cvs.server.com`, el nombre del servidor en el que está CVS. CVS ofrece una serie de comandos (se llaman con `cvs comd opciones`...) para trabajar con el sistema de revisiones, entre ellos: `checkout`, `update`, `add`, `remove`, `commit` y `diff`.

Comandos de CVS

`cvs checkout` es el comando inicial y crea su copia privada del código fuente para luego trabajar con ella sin interferir en el trabajo de otros usuarios (como mínimo se crea un subdirectorio donde estarán los archivos).

`cvs update` se debe ejecutar del árbol privado cuando hay que actualizar sus copias de archivos fuente con los cambios que otros programadores han hecho sobre los archivos del repositorio.

`cvs add file` es un comando necesario cuando hay que agregar nuevos archivos en su directorio de trabajo sobre un módulo donde ya se ha hecho previamente un *checkout*. Estos archivos se enviarán al repositorio CVS cuando se ejecute el comando `cvs commit`.

`cvs import` se puede usar para introducir archivos nuevos en el repositorio.

`cvs remove file` se utilizará para borrar archivos del repositorio (una vez que se hayan borrado del archivo privado). Este comando debe ir acompañado de un `cvs commit` para que los cambios sean efectivos, ya que se trata del comando que transforma todas las peticiones de los usuarios sobre el repositorio.

`cvs diff file` se puede utilizar sin que afecte a ninguno de los archivos implicados si se necesita verificar las diferencias entre repositorio y directorio de trabajo o entre dos versiones.

`cvs tag -R "versión"` se puede utilizar para introducir un número de versión en los archivos de un proyecto y después hacer un `cvs commit` y un proyecto `cvs checkout -r 'version'` para registrar una nueva versión.

Una característica interesante del CVS es que puede aislar cambios de los archivos aislados en una línea de trabajo separada llamada ramificación o rama (*branch*). Cuando se cambia un archivo sobre una rama, estos cambios no aparecen sobre los archivos principales o sobre otras ramas. Más tarde, estos cambios se pueden incorporar a otros ramas o al archivo principal (*merging*). Para crear una nueva rama, utilizad `cvs tag -b rel-1-0-patches` dentro del directorio de trabajo, lo cual asignará a la rama el nombre de `rel-1-0-patches`. La unión de ramas con el directorio de trabajo significa utilizar el comando `cvs update -j`. Consultad las referencias para mezclar o acceder a diferentes ramas.

Ejemplo de una sesión

Siguiendo el ejemplo de la documentación dada en las referencias, se mostrará una sesión de trabajo (en forma general) con CVS. Como CVS almacena todas los archivos en un repositorio centralizado, se asumirá que el mismo ya ha sido inicializado anteriormente. Consideremos que se está trabajando con un conjunto de archivos en C y un Makefile, por ejemplo. El compilador utilizado es gcc y el repositorio es inicializado a `gccrep`. En primer lugar, se debe obtener una copia de los archivos del repositorio a nuestra copia privada con:

```
cvs checkout gccrep
```

Esto creará un nuevo directorio llamado `gccrep` con los archivos fuente. Si se ejecuta `cd gccrep; ls`, se verá por ejemplo `CVS makefile a.c b.c c.c`, donde existe un directorio CVS que se crea para el control de la copia privada que normalmente no es necesario tocar. Después de esto, se podría utilizar un editor para modificar `a.c` e introducir cambios sustanciales en el archivo (consultad la documentación sobre múltiples usuarios concurrentes si se necesita trabajar con más de un usuario en el mismo archivo), compilar, volver a cambiar, etc. Cuando se decide que se tiene una versión nueva con todos los cambios introducidos en `a.c` (o en los archivos que sea necesario), es el momento de hacer una nueva versión almacenando `a.c` (o todos los que se han tocado) en el repositorio y hacer esta versión disponible para el resto de los usuarios:

```
cvs commit a.c
```

Utilizando el editor definido en la variable `CVSEDITOR` (o `EDITOR` si esta no está inicializada), se podrá introducir un comentario que indique qué cambios se han hecho para que sirva de ayuda a otros usuarios o para recordar qué es lo que caracterizó a esta versión y luego poder hacer un histórico.

Si se decide eliminar los archivos (porque ya se terminó con el proyecto o porque no se trabajará más con él), una forma de hacerlo es a nivel de sistema operativo (`rm -r gccrep`), pero es mejor utilizar el propio cvs fuera del directorio de trabajo (nivel inmediato superior): `cvs release -d gccrep`. El comando detectará si hay algún archivo que no ha sido enviado al repositorio y, si lo hay y se borra, preguntará si se desea continuar o no para evitar que se pierdan todos los cambios. Para mirar las diferencias, por ejemplo, si se ha modificado `b.c` y no se recuerda qué cambios se hicieron, se puede utilizar `cvs diff b.c` dentro del directorio de trabajo. Este utilizará el comando del sistema operativo `diff` para comparar la versión `b.c` con la versión que se tiene en el repositorio (siempre hay que recordar hacer un `cvs commit b.c` si se desea que estas diferencias se transfieran al repositorio como una nueva versión).

Múltiples usuarios

Cuando más de una persona trabaja en un proyecto de software con diferentes revisiones, se trata de una situación sumamente complicada porque habrá ocasiones en las que más de un usuario quiera editar el mismo fichero simultáneamente. Una posible solución es bloquear el fichero o utilizar puntos de verificación reservados (*reserved checkouts*), lo cual solo permitirá a un usuario editar el mismo fichero simultáneamente. Para ello, se deberá ejecutar el comando `cvs admin -l command` (consultad `man` para las opciones).

CVS utiliza un modelo por defecto de puntos no reservados (*unreserved checkouts*), que permite a los usuarios editar simultáneamente un fichero de su directorio de trabajo. El primero de ellos que transfiera sus cambios al repositorio lo podrá hacer sin problemas, pero los restantes recibirán un mensaje de error cuando deseen realizar la misma tarea, por lo cual, deberán utilizar comandos de cvs para transferir en primer lugar los cambios al directorio de trabajo desde el repositorio y luego actualizar el repositorio con sus propios cambios. Consultad las referencias para ver un ejemplo de aplicación y otras formas de trabajo concurrente con comunicación entre usuarios [22].

Interfaces gráficas

Contamos con un conjunto de interfaces gráficas como tkcvs* [21] desarrollada en Tcl/Tk y que soporta Subversion o la también muy popular, Cervisia [3]. En la wiki de CVS** también se puede encontrar un conjunto de clientes, *plugins* para CVS.

***http://www.twobarleycorns.net/tkcvs.html**
****http://ximbiot.com/cvs/wiki/CVS%20Clients**

1.4.3. Subversion

Como idea inicial, **Subversion** sirve para gestionar un conjunto de archivos (repositorio) y sus distintas versiones. Es interesante remarcar que no nos importa cómo se guardan, sino cómo se accede a estos ficheros y que es común utilizar una base de datos. El repositorio es como un directorio del cual se quiere recuperar un fichero de hace una semana o 10 meses a partir del estado de la base de datos, recuperar las últimas versiones y agregar una nueva. A diferencia de CVS, Subversion hace las revisiones globales del repositorio, lo cual significa que un cambio en el fichero no genera un salto de versión únicamente en ese fichero, sino en todo el repositorio, el cual suma uno a la revisión. En Debian deberemos hacer `apt-get install subversion`, si deseamos publicar los repositorios en `apache2 apt-get install apache2-common` y el módulo específico `apt-get install libapache2-subversion`.

Enlace de interés

Además del libro disponible en http://svnbook.red-bean.com/nightly/es/index.html, consultad la documentación en http://subversion.tigris.org/servlets/ProjectDocumentList.

Primer paso: crear nuestro repositorio, usuario (consideramos que el usuario es *svuser*), grupo (*svgroup*) como root hacer:

```
mkdir -p /usr/local/svn
addgroup svgroup
```

```
chown -R root.svgroup /usr/local/svn
chmod 2775 /usr/local/svn
addgroup svuser svggroup Agrego el usuario svuser al grupo svgroup.
```

Nos conectamos como *svuser* y verificamos que estamos en el grupo *svgroup* (con el comando `group`).

```
svnadmin create /usr/local/svn/pruebas
```

Este comando creará un serie de archivos y directorios para hacer el control y gestión de las versiones. Si no se tiene permiso en /usr/local/svn, se puede hacer en el directorio local:

```
mkdir -p $HOME/svndir
svnadmin create $HOME/svndir/pruebas
```

A continuación, creamos un directorio temporal:

```
mkdir -p $HOME/svntmp/pruebas
```

Nos pasamos al directorio `cd $HOME/svntmp/pruebas` y creamos un archivo, por ejemplo:

`echo Primer Archivo Svn 'date' >file1.txt` Lo trasladamos al repositorio haciendo dentro del directorio:

```
svn import file:///home/svuser/svndir/pruebas -m "Ver. Inicial"
```

Si lo hemos creado en `/usr/local/svn/pruebas`, deberíamos poner el camino completo después del archivo. El `import` copia el árbol de directorios y el `-m` permite indicarle el mensaje de versión. Si no ponemos `-m`, se abrirá un editor para hacerlo (se debe poner un mensaje para evitar problemas). El subdirectorio `$HOME/svntmp/pruebas` es una copia del trabajo en repositorio y es recomendable borrarla para no tener la tentación o cometer el error de trabajar con ella en lugar de hacerlo con el repositorio (`rm -rf $HOME/svntmp/pruebas`).

Una vez en el repositorio, se puede obtener la copia local donde podremos trabajar y luego subir las copias al repositorio. Para ello hacemos:

```
mkdir $HOME/svm-work; cd $HOME/svn-work
svn checkout file:///home/svuser/svndir/pruebas
```

donde veremos que tenemos el directorio pruebas. Se puede copiar con otro nombre agregando al final el nombre que queremos. Para añadirle un nuevo fichero:

```
cd /home/kikov/svn-work/pruebas
echo Segundo Archivo Svn 'date' >file2.txt
svn add file2.txt
svn commit -m "Nuevo archivo"
```

Es importante remarcar que una vez en la copia local (`svn-work`), no se debe indicar el camino (*path*). `svn add` marca para añadir el fichero al repositorio y realmente se añade cuando hacemos un `cvn commit`.

Nos dará algunos mensajes indicándonos que es la segunda versión. Si agregamos otra línea, la `file1.txt` con `echo 'date' >> file1.txt`, después podemos subir los cambios con: `svn commit -m "Nueva línea"`. Es posible comparar el archivo local con el del repositorio. Para hacerlo, podemos agregar una tercera línea a `file1.txt` con `echo date >> file1.txt` sin subirlo y si queremos ver las diferencias, podemos hacer: `svn diff`. Este comando nos marcará cuáles son las diferencias entre el archivo local y los del repositorio. Si lo cargamos con `svn commit -m "Nueva línea2"` (que generará otra versión), después el `svn diff` no nos dará diferencias.

También se puede utilizar el comando `svn update` dentro del directorio para actualizar la copia local. Si hay dos o más usuarios trabajando al mismo tiempo y cada uno ha hecho una copia local del repositorio y la modifica (haciendo un `commit`), cuando el segundo usuario vaya a hacer el `commit` de su copia con sus modificaciones, le dará un error de conflicto, ya que la copia en el repositorio es posterior a la copia original de este usuario (es de-

cir, ha habido cambios entremedias), con lo cual si el segundo usuario hace el `commit`, perderíamos las modificaciones del primero. Para ello deberemos hacer un `svn update` que nos indicará el archivo en conflicto y en el archivo en conflicto nos indicará cuál es el archivo y las partes del mismo que están en conflicto. El usuario deberá decidir con qué versión se queda y después podrá hacer un `commit`. Un comando interesante es el `svn log file1.txt`, que nos mostrará todos los cambios realizados en el fichero y sus correspondientes versiones.

Un aspecto interesante es que Subversion puede funcionar conjuntamente con Apache2 (y también sobre SSL) para acceder desde otra máquina o simplemente mirar el repositorio. La configuración de Apache2 y SSL se indicó en la parte de servidores, pero también se explica en Debian Administration. Para configurarlos, es necesario activar los módulos de WebDAV.

Enlace de interés

Consultad los clientes para acceder a Subversion en: http://svnbook.red-bean.com.

Enlaces de interés

Sobre la activación de los módulos de WebDAV podéis consultar la dirección http://www.debian-administration.org/articles/285 o, en su defecto, http://www.debian-administration.org/articles/208.

Como root, hacemos:

```
  mkdir /subversion
  chmod www-data:www-data Para que Apache pueda acceder al directorio
  svnadmin create /subversion Creo el repositorio
  ls -s /subversion
-rw-r-r-   1 www-data www-data   376 Sep 11 20:27 README.txt
drwxr-xr-x   2 www-data www-data   4096 Sep 11 20:27 conf
drwxr-xr-x   2 www-data www-data   4096 Sep 11 20:27 dav
drwxr-xr-x   2 www-data www-data   4096 Sep 11 20:28 db
-rw-r-r-   1 www-data www-data   2 Sep 11 20:27 format
drwxr-xr-x   2 www-data www-data   4096 Sep 11 20:27 hooks
drwxr-xr-x   2 www-data www-data   4096 Sep 11 20:27 locks
```

Para la autentificación, se utiliza `htpasswd` (por ejemplo, con la instrucción `htpasswd2 -c -m /subversion/.dav_svn.passwd user` que en nuestro ejemplo es `www-data`. La *-c* solo es la primera vez que ejecutamos el comando para crear el archivo. Esto indica que para acceder a este directorio se necesita la contraseña (que es la que hemos entrado para user). A continuación, se debe cambiar el *httpd.conf* por algo como:

```
<location /svn>
   DAV svn
   SVNPath /subversion
   AuthType Basic
   AuthName "Subversion Repository"
   AuthUserFile /subversion/.dav_svn.passwd
   Require valid-user
</location>
```

Reiniciamos Apache y ya estamos listos para importar algunos archivos, por ejemplo:

```
svn import file1.txt http://url-servidor.org/svn -m "Import Inicial"
```

Nos pedirá la autentificación (`user/passwd`) y nos dirá que el fichero `file1.txt` se ha añadido al repositorio.

1.4.4. Git

Git es un programa de control de versiones diseñado por Linus Torvalds basado en la eficiencia y la confiabilidad del mantenimiento de versiones de aplicaciones con un elevado número archivos de código fuente que, si bien se diseñó como un motor de bajo nivel sobre el cual se pudieran escribir aplica-

ciones de usuario (*front ends*), se ha convertido desde entonces en un sistema de control de versiones con funcionalidad plena a partir de su adopción como herramienta de revisión de versiones para el grupo de programación del núcleo Linux. Entre las características más relevantes encontramos:

1) Soporta el desarrollo no lineal y permite la gestión eficiente de ramificaciones y mezclado de diferentes versiones. Git incluye herramientas específicas para navegar y visualizar un historial de desarrollo no-lineal.

2) Gestión distribuida: permite a cada programador una copia local del historial del desarrollo entero y que los cambios se propaguen entre los repositorios locales.

3) Los repositorios pueden publicarse por HTTP, FTP, rsync o mediante un protocolo nativo, ya sea a través de una conexión TCP/IP simple o a través de cifrado SSH, y permite emular servidores CVS para interactuar con clientes CVS preexistentes.

4) Los repositorios Subversion y svk se pueden usar directamente con git-svn.

Instalación de un repositorio público de Git sobre Debian en 12 pasos

1) Instalamos GIT: con el apt-get (o también con Synaptic), instalamos los paquetes siguientes: `git-arch, git-core` y `gitweb` (interfaz web para git). A continuación, creamos el directorio `/var/cache/git` (el repositorio git) y también `/var/www/git` para la interfaz web: `mkdir /var/www/git` y `mkdir /var/cache/git` (verificar que no existe). Existe otro paquete llamado `git`, pero no es necesario [10].

2) Configuramos el nombre y el correo:

```
git config -global user.name "Remo"
git config -global user.email remo@debian.nteum.org
```

Podemos verificar con `git config -l`

```
user.name=Remo
user.email=remo@debian.nteum.org
```

3) Preparamos un proyecto creando el directorio en nuestro $HOME mediante la instrucción `mkdir -p projects/hello` y pasamos a él `cd projects/hello`. Aquí creamos un archivo, por ejemplo, `gedit hello.c` con el siguiente contenido (este será nuestro repositorio inicial):

```
#include <stdio.h>
int main (void)
{printf ("Hola UOC!\n");}
```

4) Creamos nuestro primer repositorio GIT:

```
remo@debian:~$ cd projects/hello
remo@debian:~/projects/hello$ git init
  Initialized empty Git repository in /home/remo/projects/hello/.git/
remo@debian:~/projects/hello$ ls .git/
  branches config description HEAD hooks info objects refs
```

Ahora deberíamos dar una descripción a nuestro proyecto haciendo: `cd .git` y modificando el archivo `description` para que contenga una línea como:

```
My first GIT project - Hello World.
```

y borrando la existente.

5) Añadimos el archivo a nuestro proyecto:

```
remo@debian:~/projects/hello$ git add hello.c
remo@debian:~/projects/hello$ git status
  # On branch master
  # Initial commit
  # Changes to be committed:
  # (use "git rm -cached <file>..." to unstage)
  # new file: hello.c
```

6) Hacemos el *commit* inicial (no olvidéis que el `-m` describe nuestro proyecto y será la información de lo que se ha hecho):

```
remo@debian:~/projects/hello$ git commit -a -m "Initial Commit"
  Created initial commit e833201: Initial Commit
   1 files changed, 5 insertions(+), 0 deletions(-)
   create mode 100644 hello.c
       remo@debian:~/projects/hello$ git status
  # On branch master
  nothing to commit (working directory clean)
```

Podemos mirar el registro con: **git log**

```
commit e8332018c780c396ee442674e420046ccb397f3b
Author: remo <remo@debian.(none)>
Date: Sun Nov 28 04:38:01 2010 -0500
 Initial Commit
```

7) Agregamos más archivos al repositorio. Para ello, editamos un archivo `gedit library.h` con el siguiente contenido:

```
#ifndef DEFINITIONS_H
#define DEFINITIONS_H 1
/* Implement a number using a linked list. */
struct LinkedListNumber
{ struct LinkedListNumber*
one_less_;
};
#endif /* DEFINITIONS_H */
```

También editamos el `hello.c` para incluir al inicio `library.h` como `#include "library.h"`
Si miramos el estado con `git status`, veremos:

```
# On branch master
# Changed but not updated: (use "git add <file>..."
to update what will be committed)
# modified: hello.c
# Untracked files: # (use "git add <file>..."
to include in what will be committed)
# library.h
no changes added to commit (use "git add.ªnd/or "git commit -a")
```

Ahora debemos agregar la biblioteca y hacer un nuevo *commit*:

```
remo@debian:~/projects/hello$ git add library.h
remo@debian:~/projects/hello$ git commit -a -m "library.h file added"
  Created commit b1042af: library.h file added
   2 files changed, 11 insertions(+), 0 deletions(-)
   create mode 100644 library.h
```

Si miramos el estado con `git status`:

```
# On branch master nothing to commit (working directory clean)
```

Y el registro con `git log`:

```
commit b1042afd6085db51eeb80586876d87c9ebd9ad36
Author: remo <remo@debian.(none)>
Date: Sun Nov 28 04:43:13 2010 -0500
 library.h file added
commit e8332018c780c396ee442674e420046ccb397f3b
Author: remo <remo@debian.(none)>
Date: Sun Nov 28 04:38:01 2010 -0500
 Initial Commit
```

Con este paso ya tenemos el primer repositorio Git y hemos realizado dos *commits*. No os preocupéis si en este momento os sentís perdidos, ya que es normal y será cuestión de trabajar y conocer los comandos para descubrir la potencia de Git.

8) Configuramos Apache y Gitweb (debe hacerse como root). Para ello, deben copiarse los archivos `gitweb.cgi`, `logo` y `css` a `/var/www/git`: `cp /usr/share/gitweb/* /var/www/git`; `cp /usr/lib/cgi-bin/gitweb.cgi /var/www/git`. Considerando que el *DocumentRoot* de Apache está en `/var/www`, se modifica la configuración de Apache creando el archivo `/etc/apache2/conf.d/git` y en su interior se pone:

```
<Directory /var/www/git>
    Allow from all
    AllowOverride all
    Order allow,deny
    Options ExecCGI
    <Files gitweb.cgi>
      SetHandler cgi-script
    </Files>
</Directory>
DirectoryIndex gitweb.cgi
SetEnv GITWEB_CONFIG /etc/gitweb.conf
```

9) Modificamos el archivo `/etc/gitweb.conf` para incluir (mirad las diferencias con el original sobre todo las / de los directorios):

```
# path to git projects (<project>.git)
$projectroot = "/var/cache/git";
# directory to use for temp files
$git_temp = "/tmp";
# target of the home link on top of all pages
#home_link =my_uri || "/";
$home_link = "/git/";
# html text to include at home page
$home_text = "indextext.html";
# file with project list; by default, simply scan the projectroot dir.
$projects_list = $projectroot;
# stylesheet to use
$stylesheet = "gitweb.css";
# logo to use
$logo = "git-logo.png";
# the 'favicon'
$favicon = "git-favicon.png";
```

Esto indica que todas nuestras copias y repositorios Git estarán en `/var/cache/git`, que es el sitio por defecto en Debian. Si lo queréis cambiar, deberéis modificar todas estas variables y la de Apache. Finalmente, se debe recargar Apache con `/etc/init.d/apache2 reload`.

10) Modificar el repositorio de trabajo. Para ello, nos cambiamos al directorio de trabajo `project` original como root:

```
debian:/home/remo/projects# git clone -bare hello hello.git
Initialized empty Git repository in /home/remo/projects/hello.git/
debian:/home/remo/projects#  ls -al
  ...
  drwxr-xr-x 3 remo remo 4096 2010-11-28 04:40 hello
  drwxr-xr-x 7 root root 4096 2010-11-28 05:10 hello.git
```

Verificamos que existe el servicio: `cat /etc/services | grep ĝit`, que nos deberá dar: `git 9418/tcp # Git Version Control System` y hacemos que el directorio hello.git sea exportable: `touch hello.git/git-daemon-export-ok`.

11) Ahora movemos el directorio a su sitio definitivo (al sitio indicado por la variable de `/etc/gitweb.conf`):

```
debian:/home/remo/projects# mv hello.git /var/cache/git/
```

Y finalmente ya podéis ver vuestro proyecto en `http://localhost/git/`, que dará una (salida) navegable como:

```
projects/hello.git/
summary | shortlog | log | commit | commitdiff | tree
  description My first Git Repository: hello.c + library.h
  owner root
  last change Sun, 28 Nov 2010 09:43:13 +0000
Shortlog
2 hours ago remo library.h file added master commit | commitdiff | tree | snapshot
2 hours ago remo Initial Commit commit | commitdiff | tree | snapshot
heads
2 hours ago master shortlog | log | tree
My first Git Repository: hello.c + library.h
RSS Atom
```

12) Finalmente, como todo estará en web es necesario borrar el repositorio original para continuar trabajando de un clon del repositorio web :

```
remo@debian:~/projects$ rm -rf hello
remo@debian:~/projects$ git clone /var/cache/git/hello.git/ hello
  Initialized empty Git repository in /home/remo/projects/hello/.git/
remo@debian:~/projects$ ls hello
  hello.c library.h
```

Como se puede ver, estos archivos tienen todas las modificaciones. Por último, consultad la documentación para clonar y hacer cambios desde otra máquina.

1.4.5. Mantis Bug Tracker

Mantis Bug Tracker es un sistema (GPL2) de seguimiento de errores a través de la web. El uso más común de MantisBT es hacer un seguimiento de errores o incidencias, pero también sirve como herramienta de seguimiento de gestión y administración de proyectos. Desde 2008, el proyecto ha cambiado el programa de seguimiento de proyectos de Subversion a GIT y en estas fechas (noviembre de 2010), se puede integrar con Gitweb, GitHub, WebSVN, SourceForge (*open-source software hosting facility*; únicamente para integraciones de Subversion). Además del paquete de Mantis, es necesario tener instalado MySQL, Apache Web Server y el módulo PHP para Apache. Para verificar que MysSQL funciona correctamente, se puede utilizar phpMyAdmin (`http://localhost/phpmyadmin/`) y para comprobar que también se tienen instalados los módulos PHP en Apache, se puede crear un archivo `/var/www/prueba.php` con el siguiente contenido:

```
<html>
<body>
<h1>Prueba de PHP y Apache. </h1>
<?php phpinfo() ;?>
</body>
</html>
```

Accediendo a `http://localhost/prueba.php`, se podrá ver toda la información (un gran conjunto de tablas en colores de la gama del azul) con la información de php. A partir de aquí, se puede instalar Mantis con `apt-get install mantis` o desde Synaptic. Durante la instalación, nos indicará que la contraseña para el usuario administrator es root (y el aviso de que es recomendable cambiarla) y nos solicitará si queremos hacer la integración con MySQL a través del paquete `dbconfig-common`. Es recomendable hacerlo de esta forma y nos pedirá el usuario y la contraseña para acceder a MySQL. A partir de ese momento, nos podremos conectar a `http://localhost/mantis/` introduciendo el usuario administrator y la contraseña root (se recomienda que la primera acción sea cambiar la contraseña en la sección `MyAccount`). Accediendo desde la interfaz web, se podrán crear nuevos usuarios con permisos de usuario por roles (administrator, viewer, reporter, updater, etc.), definir los proyectos y las categorías dentro de cada proyecto, gestionar los anuncios, informes, registros, tener una visión general de los proyectos y su estado, gestionar los documentos asociados, etc.

Hay un amplio conjunto de opciones que se pueden modificar en el archivo `/usr/share/mantis/www/config_inc.php` [14], así, para cambiar la lengua del entorno, se edita y añade `$g_language_choices_arr = array( 'english', 'spanish'); $g_default_language = 'spanish';` y para que la página principal del Mantis Bug Tracker sea automáticamente el propio Bug Tracker y se permita un acceso anónimo a los proyectos públicos:

1) Crear una cuenta de usuario, por ejemplo anonymous o guest, dejando en blanco el `Real Name`, `Email=anonymous@localhost` (o dejar en blan-

co si se pone `$g_allow_blank_email = ON`), `Access Level = viewer` o `reporter` (dependiendo los que se quiera) `Enabled = true` y `Protected = true`.

2) Después se tiene que modificar en el archivo anterior (`config_inc.php`) poniendo las siguientes variables:

```
$g_allow_anonymous_login = ON;
$g_anonymous_account = 'anonymous'
```

y, opcionalmente, para dejar en blanco las direcciones de correo:

```
$g_allow_blank_email = ON.
```

Para más configuraciones o integraciones con otros paquetes, consultad el manual [8] o acceded a la página de la *wiki* del proyecto* [14]. Se puede ejecutar una demo en los propios servidores del proyecto**.

***http://www.mantisbt.org /wiki/doku.php /mantisbt:mantis_recipies**
****http://demo.mantisbt.org /my_view_page.php**

Actividades

1. Definid en PostgreSQL una base de datos que tenga al menos 3 tablas con 5 columnas (de las cuales 3 deben ser numéricas) en cada tabla. Generad un listado ordenado por cada tabla/columna. Generad un listado ordenado por el mayor valor de la columna X de todas las tablas. Cambiad el valor numérico de la columna Y con el valor numérico de la columna Z + el valor de la columna W/2.

2. Realizad el mismo ejercicio anterior, pero con MySQL.

3. Configurad el CVS para hacer tres revisiones de un directorio donde hay 4 archivos .c y un Makefile. Haced una ramificación (*branch*) de un archivo y luego mezcladlo con el principal.

4. Simulad la utilización de un archivo concurrente con dos terminales de Linux e indicar la secuencia de pasos para hacer que dos modificaciones alternas de cada usuario queden reflejadas sobre el repositorio CVS.

5. Realizad el mismo ejercicio anterior, pero uno de los usuarios debe conectarse al repositorio desde otra máquina.

6. Realizad nuevamente los tres ejercicios anteriores, pero en Subversion.

7. Cread un repositorio sobre Git y hacedlo visible a través del web, de tal modo que dos usuarios de la misma máquina puedan modificar los archivos y actualizar el repositorio.

8. Instalad Mantis, generad un usuario anónimo que pueda acceder a un proyecto público como reporter, pero no a uno privado.

Bibliografía

[1] *Apache2 y SSL*. <http://www.debian-administration.org/articles/349>

[2] **Cederqvist**. *Version Management with CVS*.
<http://www.cvshome.org>

[3] **Cervisia**. *Interfaz Cervisia para CVS*.
<http://cervisia.kde.org/>

[4] **Debian.org**. *Debian Home*.
<http://www.debian.org>

[5] **Ibiblio.org** (2010). *Linux Documentation Center*.
<http://www.ibiblio.org/pub/Linux/docs/HOWTO/>

[6] **Mourani, G.** (2001). *Securing and Optimizing Linux: The Ultimate Solution*. Open Network Architecture, Inc.

[7] *Documentación de MySQL*.
<http://dev.mysql.com/usingmysql/get_started.html>

[8] *Documentación del proyecto Mantis*.
<http://www.mantisbt.org/documentation.php>

[9] *Git. Fast Control Version system*.
<http://git-scm.com/>

[10] *Instalar Git sobre Debian*.
<http://linux.koolsolutions.com/2009/08/07/learn-git-series-part-1-installing-git-on-debian/>

[11] *Integración WebDAv con Subversion*.
<http://www.debian-administration.org/articles/285>
<http://www.debian-administration.org/articles/208>

[12] **Kiesling, R.** *The RCS (Revision Control System)*. The Linux Documentation Project.

[13] *Libro sobre Subversion*.
<http://svnbook.red-bean.com/nightly/es/index.html>
(versión en español <http://subversion.tigris.org/servlets/ProjectDocumentList>)

[14] *Mantis Bug Tracker Wiki*.
<http://www.mantisbt.org/wiki/doku.php>

[15] *Módulos de Webmin*.
<http://doxfer.webmin.com/Webmin>

[16] *Mysql Administrator*.
<http://www.mysql.com/products/tools/administrator/>

[17] *PgAdmin*.
<http://www.pgadmin.org>

[18] *PgpPGAdmin*.
<http://phppgadmin.sourceforge.net/>

[19] *PostgresSQL*.
<http://www.postgresql.org>

[20] *Subversion*.
<http://subversion.apache.org/>

[21] **Tkcvs**. *Interfaz Tkcvs para CVS*.
<http://www.twobarleycorns.net/tkcvs.html>

[22] **Vasudevan, A.** *CVS-RCS (Source Code Control System)*.

[23] *WebMin*.
<http://www.webmin.com/>