Administración de datos

Remo Suppi Boldrito

PID_00212474

UOC
Universitat Oberta de Catalunya

Índice

Introducción

Según afirman algunos expertos, entre 2013-2018 se generarán tantos dados como en los últimos 5000 años. Este fenómeno conocido como *"Big Data"* es utilizado para referirse a situaciones donde el conjunto de datos a tratar supera las medidas habituales de datos gestionados hoy en día por los sistemas de información y que deberán ser obtenidos, almacenados y procesados en tiempos razonables. ¿Cómo se considera *Big Data*? Es un criterio dinámico, pero normalmente se puede considerar como un conjunto de datos único de entre decenas de *Terabytes* a decenas de *PetaBytes* (10 PBytes = 10.000 TBytes = 10 millones de Gigabytes) y donde las dificultades que se encontrarán estarán en su captura, almacenado, búsqueda, compartición, lectura, análisis, y visualización. Vinculadas a esta nueva tendencia, han recuperado fuerza tecnologías conocidas como *Data Minining* (Minería de Datos) en su versión más enfocada al procesamiento de datos sociales (*Social Data Mining*) o el *Datawarehouse* (Repositorio de Datos), que serán aspectos predominantes a tener en cuenta en el futuro cercano.[1]

Es por ello por lo que la gestión de datos es un aspecto muy importante en la utilización de la tecnología actual, y los sistemas operativos son la base donde se ejecutarán las aplicaciones que capturarán, almacenarán y gestionarán estos datos. Un manejo eficiente en el almacenamiento, gestión y procesamiento de los datos no puede verse, hoy en día, separado de una **Base de Datos** (*databases*, DB) y que serán el punto crítico para las nuevas tecnologías que se desarrollen a partir del *Big Data*. Una base de datos es un conjunto estructurado de datos que pueden organizarse de manera simple y eficiente por parte de 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 (ejemplos de ellas son MySQL/MariaDB, PostgreSQL). 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.

También en necesario destacar que comienzan, con el tratamiento derivado del *BigData*, a desarrollarse tecnologías para gestionar datos no estructurados, en lo que se denominan bases de datos no estructuradas (o NoSQL o también "no solo SQL") y que definen sistemas de gestión de bases de datos que difieren de las tradicionales BD relacionales (RDBMS), en las que, por ejemplo, no usan SQL como el principal lenguaje de consultas, ya que no se realizan búsquedas exactas, la operación más típica es la búsqueda por similitud, los datos almacenados no requieren estructuras fijas como tablas y si utilizan categorías

como por ejemplo clave-valor. Las bases de datos NoSQL han crecido con el auge de grandes compañías de Internet (Google, Amazon, Twitter y Facebook entre otros), ya que necesitan encontrar formas de tratamiento de los datos producidos y que con las RDBMS no pueden o es muy complejo/ineficiente y donde el rendimiento/eficiencia en su procesamiento y sus propiedades de tiempo real son más importantes que la coherencia. Ejemplo de estas bases de datos son Hadoop-Hbase (utilizada por casi todas las grandes compañías de Internet), MongoDB (NoSQL orientado a documentos), Elasticsearch (*multitenancy*, *full-text search engine* con interfaz web RESTful y soporte para *schema-free JSON documents*) o Redis (motor de base de datos en memoria, basado en el almacenamiento en tablas de *hashes* -clave/valor-).

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 PostgreSQL y MySQL (o su *fork* MariaDB), que serán objeto de nuestro análisis.

También veremos algunos aspectos introductorios sobre SQLite, que es un sistema de gestión de bases de datos relacional (y compatible con ACID) contenido en una biblioteca (escrita en C) y que, a diferencia de los sistemas anteriores (cliente-servidor), el motor de SQLite no es un proceso independiente que se comunica con el programa sino que SQLite se enlaza con el programa pasando a ser parte del mismo. El programa utiliza la funcionalidad de SQLite a través de llamadas simples a funciones, reduciendo el acceso a la base de datos y siendo mucho más eficientes. El conjunto de la base de datos (definiciones, tablas, índices, y los propios datos) son guardados en un solo archivo estándar en la máquina host y la coherencia de la BD se logra bloqueando todo el fichero de base de datos al principio de cada transacción.

Por otro lado, 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, Bazaar, Darcs, Mercurial, Monotone, Codeville, RCS, etc.).

En este módulo veremos **CVS (Control Version System)**, **Subversion**, **GIT** y **Mercurial** 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.

A continuación veremos dos grandes entornos en la gestión de repositorios dadas sus prestaciones y utilización en grandes proyectos: GIT y Mercurial. **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. **Mercurial** es un sistema de control de versiones (básicamente desarrollado en Python), y diseñado para obtener el mejor rendimiento y escalabilidad, con un desarrollo completamente distribuido (sin necesidad de un servidor) y que permite una gestión robusta de archivos (texto o binarios) y con capacidades avanzadas de ramificación e integración. 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 (generalmente MySQL y Apache).

1.1. PostgreSQL

En el lenguaje de bases de datos, PostgreSQL utiliza un modelo cliente-servidor [3]. 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, ya que 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`.[3]

Instalación paso a paso

En este apartado se optará por la instalación y configuración de los paquetes de la distribución (tanto de la versión distribuida como la de la última versión de desarrollador) pero su instalación desde los fuentes no genera dificultad y es la adecuada si se desea tener la última versión de la BD. Los fuentes pueden obtenerse de http://www.postgresql.org/ftp/source/ y siguiendo las indicaciones desde http://www.postgresql.org/docs/9.3/interactive/installation.html.

La instalación es muy simple, ejecutando `apt-get install postgresql` instalará la versión 9.1 de la distribución de Debian Wheezy (junto con la instalación de otros paquetes adicionales como postgresql-client-9.1 postgresql-client-common postgresql-common). En el caso de que se desee instalar la última versión (9.3) desde los paquetes binarios, PostgreSQL facilita los paquetes para Debian (http://www.postgresql.org/download/linux/debian/) y se debe agregar el repositorio, p. ej., creando un archivo */etc/apt/sources.list.d/pgdg.list* que tenga la línea `deb http://apt.postgresql.org/pub/repos/apt/ wheezy-pgdg main`; luego debemos importar la llave del repositorio con

```
wget -quiet -O - https://www.postgresql.org/media/keys/ACCC4CF8.asc
| apt-key add -
```

y actualizar el repositorio `apt-get update`, finalmente podremos ejecutar `apt-get install postgresql-9.3`. Si se desea instalar el paquete que incluye diferentes contribuciones de la comunidad* deberemos hacer `apt-get install postgresql-contrib` (estas contribuciones incluyen *fuzzystrmatch, unaccent o citext* para búsquedas intensivas o difusas por ejemplo). Después de la instalación tendremos: cliente postgresql (`psql`), usuario del sistema por defecto: Debian ha creado el usuario **postgres** (no cambiar el passwd de este usuario), usuario por defecto de postgresql: es el *superuser* **postgres** (su passwd deberá ser cambiado desde dentro de la BD), usuario de la BD **postgres**, cluster por defecto: **main**, BD por defecto: **template1**, schema por defecto: **public**. Para cambiar el *passwd* del usuario **postgres** de la BD hacemos: `su -l root` a continuación `su - postgres` y luego `psql` para entrar en la consola (cliente o frontend). Después de los mensajes y delante del prompt *postgres=#* introducimos `\password postgres` y el *passwd* seleccionado dos veces. Para salir hacemos `\q`. A continuación se deberá hacer algunos ajustes como del archivo */etc/postgresql/9.1/main/postgresql.conf* cambiar la línea *# listen_addresses = 'localhost'* por *listen_addresses = 'localhost, 192.168.1.200'* donde la IP es la de servidor. También definir el método de autentificación en */etc/postgresql/9.1/main/pg_hba.conf* agregar por ejemplo

***http://www.postgresql.org /docs/9.3/static /contrib.html**

host all all 192.168.1.200/24 md5
host all all 192.168.1.100/24 md5

Donde 192.168.1.200 es la dirección de servidor y 192.168.1.100 la IP de gestión de la DB y deberemos activar, si lo tenemos instalado, `iptables` para aceptar comunicaciones en el puerto 5432. Finalmente deberemos reiniciar el servidor con `service postgresql restart` para que los cambios sean efectivos. Los archivos esenciales de PostgreSQL se encuentran en: configuración */etc/postgresql/9.1/main/*, binarios */usr/lib/postgresql/9.1/bin*, archivos de datos */var/lib/postgresql/9.1/main* y logs */var/log/postgresql/postgresql-9.1-main.log*. Para ver los *clusters* disponibles (PostgreSQL se basa en CLUSTERS que es un grupo de BD manejadas por un servicio común y donde cada una de estas BD, contiene uno o más SCHEMATA) podemos ejecutar `pg_lsclusters` que en nuestro caso es:

Version Cluster Port Status Owner Data directory Log file
9.1 main 5432 online postgres /var/lib/postgresql/9.1/main /var/log/postgresql/postgresql-9.1-main.log

Podéis ver la documentación de PostgreSQL en la dirección web siguiente: http://www.postgresql.org/docs/9.1/interactive/index.html.

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. Con el comando `!l` nos listará todas las BD definidas.

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 apartado 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` (o previamente hacer `su -l root`), 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/lib/postgresql/9.1/bin/createdb nteumdb`), donde la ruta dependerá de la instalación que se haya hecho (consultar referencias para la solución de problemas). Otros mensajes de error serían *could not connect to server*, cuando el servidor no está arrancado, o también el mensaje *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:

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*).

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 apartado de instalación.

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 =#` o `nteumdb =>` (el primero si es el superusuario y el segundo si es un usuario normal). 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. Una lista de los comandos SQL que permite la podemos consultar en la dirección web http://www.postgresql.org/docs/9.1/static/sql-commands.html.

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 correspondencia, 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 SUPERUSER;`
- **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 hay la posibilidad de 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`, donde este último comando 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 estrategia es guardar los archivos de las bases de datos a nivel del sistema operativo, p. ej. con `tar -cf backup.tar /var/lib/postgresql/9.1/main`. 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.

1.2. 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 distribución de los fuentes 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, ejecutad `\i basics.sql` dentro. El parámetro `\i` lee los comandos del archivo especificado (`basic.sql` en nuestro caso). PostgreSQL, como ya hemos mencionado, 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/postgresql/backups/dumpall.sql`
Para restaurar una base de datos: `# su - postgres` y después
`$ psql -f /var/lib/postgresql/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`

Para un usuario que se inicia en PostgreSQL es interesante obtener una base de datos ya construida y practicar con ella. Por ejemplo, en la dirección web

http://pgfoundry.org/projects/dbsamples/ podemos obtener la base World 1.0, que nos da una lista de 4079 ciudades del mundo de 239 países y su población. Una vez descargada (formato tar.gz) la descomprimimos con `tar xzvf world-1.0.tar.gz` y entramos en `psql`, desde el prompt hacemos `\i /tmp/world.sql`. Veremos cómo se crean las tablas y ya estaremos en condiciones de hacer consultas como:

```
SELECT * FROM "pg_tables" WHERE schemaname = 'public';     --Miramos las tablas
 schemaname |    tablename    | tableowner | tablespace | hasindexes | hasrules | hastriggers
------------+-----------------+------------+------------+------------+----------+-------------
 public     | city            | postgres   |            | t          | f        | t
 public     | country         | postgres   |            | t          | f        | t
 public     | countrylanguage | postgres   |            | t          | f        | t

SELECT * FROM "city" WHERE countrycode = 'ESP';         --Miramos las ciudades de un país
 id  |              name              | countrycode |      district      | population
-----+--------------------------------+-------------+--------------------+------------
 653 | Madrid                         | ESP         | Madrid             |    2879052
 654 | Barcelona                      | ESP         | Katalonia          |    1503451
 655 | Valencia                       | ESP         | Valencia           |     739412
...

SELECT count(*) FROM "city" WHERE countrycode = 'ESP';      --Contamos las ciudades de un país
 count
-------
    59

SELECT * FROM "city" WHERE name = 'Barcelona';              --Miramos las ciudades con un nombre
  id  |   name    | countrycode |  district  | population
------+-----------+-------------+------------+------------
  654 | Barcelona | ESP         | Katalonia  |    1503451
 3546 | Barcelona | VEN         | Anzoátegui |     322267

SELECT * FROM "city" WHERE name = 'Barcelona' and countrycode = 'ESP'; --Y solo las de un país
 id  |   name    | countrycode | district  | population
-----+-----------+-------------+-----------+------------
 654 | Barcelona | ESP         | Katalonia |    1503451

SELECT * FROM "city" WHERE population > '9500000';   --Miramos las ciudades con la población mayor
  id  |      name       | countrycode |   district    | population
------+-----------------+-------------+---------------+------------
  206 | Sao Paulo       | BRA         | Sao Paulo     |    9968485
  939 | Jakarta         | IDN         | Jakarta Raya  |    9604900
 1024 | Mumbai (Bombay) | IND         | Maharashtra   |   10500000
 1890 | Shanghai        | CHN         | Shanghai      |    9696300
 2331 | Seoul           | KOR         | Seoul         |    9981619

SELECT TRUNC(AVG(population),1) AS Total  FROM "city" WHERE population > '9500000';

                            --Obtenemos la media de las ciudades más pobladas
   total
------------
 9950260.8
```

Para más detalles sobre PostgreSQL y su utilización recomendamos [2][4].

1.2.1. Entornos de administración gráficos

Existen diversos entornos de administración gráficos, como el que hemos instalado en el apartado anterior, **Phppgadmin** y **Pgadmin** (disponible en la mayoría de las distribuciones, Debian incluida). Estos programas permiten ac-

ceder y administrar una base de datos con una interfaz gráfica ya sea mediante web o por interfaz propia. En el caso de interfaz propia, 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.

Para instalar el paquete PhpPgAdmin deberemos ejecutar `apt-get install phppgadmin` y estará disponible en la URL *http://localhost/phppgadmin*. Para tenerlo disponible desde la dirección del propio servidor (o desde otra) deberemos modificar el archivo */etc/apache2/conf.d/phppgadmin* y comentar la línea `allow from 127.0.0.0/255.0.0.0 ::1/128` y descomentar la línea `allow from all` reiniciando el servidor luego. Cuando deseemos entrar en la base de datos en el menú de la izquierda nos pedirá un usuario/*passwd* que si introducimos "postgres" y el *passwd* del superusuario nos dará un error diciendo *Login disallowed for security reasons*, para habilitar su acceso iremos al archivo */etc/phppgadmin/config.inc.php* y cambiaremos la línea

```
$conf['extra_login_security'] = true;
```

por `$conf['extra_login_security'] = false;` a continuación deberemos recargar la página y reiterar el procedimiento y ya podremos acceder a la gestión de la base de datos.

Si se desea evitar trabajar con el usuario "postgres" (es recomendable) se puede crear un usuario con contraseña para acceder a la base de datos, para ello debemos hacer desde dentro de `psql`: `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 creados anteriormente. A partir de aquí, veremos las bases de datos, así como todos los parámetros y configuraciones.

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. La instalación para Debian se detalla en http://www.webmin.com/deb.html (Webmin fue retirado del repositorio Debian/Ubuntu en 2006 por retardos en la solución de errores y problemas en la política de mantenimiento del paquete https://bugs.debian.org/cgi-bin/bugreport.cgi?bug=343897).

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 y utilizada por grandes compañías de Internet (Facebook, Google, Adobe, Twiter, Youtube, Zappos, entre otros...). La compañía que desarrolla y mantiene la BD y sus utilidades se denomina MySQL AB (en 2008 esta fue adquirida por Sun Microsystems y a su vez esta lo fue en 2010 por Oracle Corporation, por lo cual MySQL es subsidiaria de Oracle Co.) y desarrolla MySQL como software libre en un esquema de licencia dual. Por un lado se ofrece bajo la GNU GPL para cualquier uso compatible con esta licencia (versión llamada MySQL Community Edition), pero para aquellas empresas que quieran incorporarlo en productos privados, deben comprar una licencia específica que les permita este uso (versiones MySQL Enterprise Edition y MySQL Cluster CGE básicamente). Es interesante analizar las diferentes situaciones (sobre todo en nuestra orientación hacia el Open Source) de licencias de MySQL. En la opción GPL, podemos utilizar libremente MySQL, es decir, sin coste alguno cuando la aplicación se desarrolla a nivel local y no se utiliza comercialmente. Por ejemplo, MySQL se pueden usar libremente dentro de un sitio web (si se desarrolla una aplicación PHP y se instala en un proveedor de servicios de Internet, tampoco se tiene que hacer que el código PHP esté disponible libremente en el sentido de la GPL).

Enlace de interés

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

Igualmente, un proveedor de servicios de Internet puede hacer MySQL disponible a sus clientes sin tener que pagar licencia de MySQL (siempre y cuando MySQL se ejecute exclusivamente en el equipo del ISP) o MySQL se puede utilizar de forma gratuita para todos los proyectos GPL o licencia libre comparable (por ejemplo, si hemos desarrollado un cliente de correo electrónico Open Source para GNU/Linux, y se desea almacenar datos de los correos en una base de datos MySQL) y todos los desarrollos/extensiones que se realicen sobre los productos MySQL debe ser abiertos/publicados y no comerciales. El uso de MySQL con una licencia comercial se aplica cuando desarrollamos productos comerciales y no están abiertos/publicados; es decir. no se puede desarrollar un producto comercial (p. ej.,. un programa de contabilidad con MySQL como base de datos) sin que el código esté disponible como código abierto. Si las limitaciones de la GPL no son aceptables para quien desarrolla la aplicación entonces se puede vender el producto (programa), junto con una licencia de MySQL comercial (más información en http://www.mysql.com/about/legal/). Considerando la licencia GPL, se puede obtener el código fuente, estudiarlo y modificarlo de acuerdo con sus necesidades sin pago alguno, pero estaremos sujetos a los deberes que impone GPL y deberemos publicar como código abierto lo que hemos desarrollado también. 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.

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.

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-5.5`, `server-5.5` y `common` son necesarios). La instalación, después de unas preguntas, creará un usuario que será el superusuario de la BD (p.ej., admin) 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 (que no sean Debian ni RPM, para ellos 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');
mysql>select user,host,password from mysql.user;
```
Muestra los usuarios, host de conexión, y passwd (también se pueden poner los comandos en minúsculas).
```
mysql>delete from mysql.user where user="";
```
Borra los usuarios anónimos (sin nombre en la info anterior).

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: 5.5.35-0+wheezy1 (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 Workbench***, que es una aplicación potente para el diseño administración y control de bases de datos basadas en MySQL (incluye el *Database Administration* que reemplaza el anterior MySQL Administrator). Esta aplicación es un conjunto de herramientas para los desarrolladores y administradores de BD que integra el desarrollo, gestión, control y el mantenimiento de forma simple y en un mismo entorno de la BD. Las partes principales son: diseño y modelado de BD, desarrollo en SQL (reemplaza el MySQL Query Browser), administración de la BD (reemplaza MySQL Administrator) y migración de BD.

Enlace de interés

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

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

Para usuarios recién iniciados (o expertos) recomendamos, por su facilidad en el uso y simplicidad, **PhpMyAdmin** (incluida en la mayoría de las distribuciones incluida Debian), que es una herramienta de software libre escrito en PHP para gestionar la administración de MySQL a través de la web. 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/aptitude`, etc.) que durante el proceso de instalación pedirá con qué servidores de web se quiere hacer la integración y el usuario de la base de datos. 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 la que ya hemos comentado en la sección de PostgreSQL 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 (p. ej., Debian|Ubuntu), se puede descargar desde http://www.webmin.com. Durante la instalación, el Webmin avisará de que el usuario principal será el root y usará la misma contraseña que el root del sistema operativo. Será posible conectarse, p. ej., 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. MariaDB

MariaDB[9] es un sistema de gestión de bases de datos derivado de MySQL con licencia GPL que incorpora dos mecanismos de almacenamiento nuevos: Aria (que reemplaza con amplias ventajas a MyISAM) y XtraDB (que sustituye InnoDB el actual del Mysql). Como *fork* de MySQL mantiene una alta compatibilidad ya que posee los mismos comandos, interfaces, API y librerías con el objetivo que se pueda cambiar un servidor por otro. La motivación de los desarrolladores de hacer un *fork* de MySQL fue la compra de Sun Microsystems por parte de Oracle en el 2010 (Sun previamente había comprado MySQL AB que es la compañía que desarrolla y mantiene MySQL) y con el objetivo de mantener esta sobre GPL ya que estaban/están convencidos que el único interés de Oracle en MySQL era reducir la competencia que MySQL daba a su producto comercial (Oracle DB).

MariaDB es equivalente a las mismas versiones de MySQL (MySQL 5.5 = MariaDB 5.5, no obstante, la versión actual es MariaDB10.1) y existen una serie de ventajas con relación a MySQL que resumimos a continuación*:

***https://mariadb.com/kb/en/mariadb/mariadb-vs-mysql-features/**

1) Mecanismos de almacenamiento: Además de los estándar (MyISAM, Blackhole, CSV, Memory y Archive) se incluyen Aria (alternativa a MyISAM resistente a caídas), XtraDB (reemplazo directo de InnoDB), FederatedX (reemplazo directo de Federated), OQGRAPH, SphinxSE, Cassandra (NoSQL y en MariaDB 10.0) TokuDB (desde MariaDB 5.5) y se están trabajando en otras NoSQL, y las últimas CONNECT, SEQUENCE y Spider (solo a partir de MariaDB 10.0)

2) Facilidad de uso y velocidad: incluye nuevas tablas y opciones para generar estadísticas de índices y tablas, procesos, gestión del tiempo en microsegundos, características NoSQL (p. ej.,. HandlerSocket), columnas dinámicas y subqueries. Con relación a la velocidad de procesamiento se pueden ver las comparaciones con relación a MySQL en https://mariadb.com/blog/mariadb-53-optimizer-benchmark.

3) Test, errores y alertas: incluye un extenso conjunto de test en la distribución que permiten analizar diferentes combinaciones de configuración y sistema operativo. Estos tests han permitido reducir notablemente los errores y alertas por lo cual se traduce en una BD altamente estable y segura con relación a la ejecución y funcionamiento.

4) Licencia: TODO el código en MariaDB está bajo GPL, LPGL o BSD y no dispone de módulos cerrados como MySQL Enterprise Ed. (De hecho todo el código cerrado en MySQL 5.5 E.Ed. está en MariaDB como *open source version*). MariaDB incluye test para todos los errores solucionados, mientras que Oracle no hace lo mismo con MySQL 5.5 y todos los errores y planes de desarrollos son públicos y los desarrolladores pertenecen a la comunidad.

Es importante tener en cuenta que MariaDB es desarrollada por la comunidad y la responsabilidad de desarrollo y mantenimiento corren a cargo de SkySQL Corporation Ab, empresa fundada en 2010 por la mismas personas de MySQL AB (David Axmark, Michael 'Monty' Widenius y Kaj Arnö). Esta empresa se fusionó con el Monty Program en 2013 (programa fundado por Monty Widenius -desarrollador de MySQL junto con D. Axmark- después que vendió la compañía a Sun MS y decidió escribir MariaBD). En 2014 SkySQL crea otra marca llamada MariaDB AB y ofrece MariaDB Enterprise, MariaDB Enterprise Cluster y MaxScale bajo un modelo diferente de negocio que la licencia de software como MySQL, para dirigirse a un mercado empresarial garantizando fiabilidad y confiabilidad en aplicaciones de misión crítica y alto rendimiento.*https://mariadb.com/about*

1.4.1. Instalación MariaDB sobre Debian

La instalación es sumamente sencilla y tal como se indica en [10] (los paquetes estarán en el repositorio Debian a partir de la próxima version Jessie). Se selecciona la distribución, versión del SO y repositorio, y se ejecuta:

```
Si tenemos BD en MySQL hacer una copia de resguardo parando los servicios primero y luego volcando la BD:
        service apache2 stop; service mysql stop
        mysqldump -u root -p --all-databases > mysqlbackup.sql
Configuramos el repositorio:
        apt-get install python-software-properties
        apt-key adv --recv-keys --keyserver keyserver.ubuntu.com 0xcbcb082a1bb943db
        add-apt-repository 'deb http://mirror.vpsfree.cz/mariadb/repo/10.1/debian wheezy main'
Luego de importada la llave actualizamos el repositorio e instalamos:
        apt-get update
        apt-get install mariadb-server
```

```
        service apache2 start
Podremos mirar la versión con: mysql  --version
        mysql  Ver 15.1 Distrib 10.1.0-MariaDB, for debian-linux-gnu (x86_64) using readline 5.1
O las bases de datos existentes con:
        mysql -u root -p -Be 'show databases'
Y si accedemos a phpmyadmin veremos en "Home"
        MySQL
        Server: Localhost via UNIX socket
        Server version: 10.1.0-MariaDB-1~wheezy
        Protocol version: 10
        User: root@localhost
        MySQL charset: UTF-8 Unicode (utf8)
```

Veremos que si tenemos instalado `mysql-server` lo desinstala y lo reemplaza por `mariadb`, haciendo las mismas preguntas de las bases de datos creadas (usuario y *passwd*).

1.5. SQLite

Como ya dijimos en la introducción, SQLite es un sistema de gestión de bases de datos relacional compatible con ACID, contenida en una librería (escrita en C) bajo licencia *Public Domain* (http://www.sqlite.org/copyright.html). Como gran diferencia de los sistemas de gestión de bases de datos cliente-servidor, el motor de SQLite no es un proceso independiente del programa que desea realizar un acceso a la base de datos, sino una librería que se enlaza junto con el programa y pasa a ser parte del mismo como cualquier otra librería (p. ej., igual que la */lib/x86_64-linux-gnu/libc.so.6* en Debian para un programa en C). El programa que necesita la funcionalidad de SQLite simplemente llama subrutinas y funciones teniendo como ventaja substancial la reducción de la latencia de acceso a la base de datos ya que las llamadas a funciones son más eficientes que la comunicación entre procesos. El conjunto de la base de datos (definiciones, tablas, índices, y los propios datos), son guardados en un archivo fichero sobre la máquina *host* y la coherencia en las transacciones se obtiene bloqueando todo el fichero al principio de cada transacción. La biblioteca implementa la mayor parte del estándar SQL-92, y mantiene las características necesarias para que una serie de instrucciones puedan ser consideradas como una transacción (ACID compliant) y su diseño permite que varios procesos o *threads* puedan acceder a la misma base de datos sin problemas donde los accesos de lectura pueden ser servidos en paralelo y un acceso de escritura solo puede ser servido si no se está haciendo ningún otro acceso en forma concurrente (lo dará un error o podrá reintentarse durante un cierto *timeout* programable).[11]

En Debian se incluye un paquete que implementa una interfaz en línea de comandos para SQlite2/3 llamado `sqlite` (versión2) o `sqlite3` (`apt-get install sqlite3`) para acceder y modificar BD SQLite además de un conjunto de librerías que permiten la interacción entre el lenguaje y la aplicación, por ejemplo, python-sqlite, ruby-sqlite, php5-sqlite como interfaces para SQLite desde phyton, ruby y php respectivamente. Además existen otras herra-

mientas para diseñar, crear o editar una base de datos compatible con SQLite, como por ejemplo SQLite Database Browser (GPL) (`apt-get install sqlitebrowser`) que permite en forma simple y visual acceder a los ficheros que contiene la base de datos [12]. Entre los principales usuarios de SQLite existen (http://www.sqlite.org/famous.html): Adobe (Photoshop Lightroom, AIR), Airbus (flight software A350), Apple (OSX, iPhone, iPod), Dropbox, Firefox, Thunderbird, Google (Android, Chrome), PHP, Python, Skype, Tcl/Tk entre otros.

Para instalar la librería SQLite en Debian, ejecutamos la instrucción `apt-get install libsqlite0-dev` (que incluye la librería y los archivos .h para desarrollar aplicaciones en C, si solo necesitamos la librería con *libsqlite0* hay suficiente). Para visualizar por ejemplo una de las bases de datos de Firefox hacemos:

```
Buscamos nuestro profile de firefox/iceweasel en el usuario de login: cd .mozilla/firefox/*.default
Abrimos, por ejemplo, la base de datos: sqlite3 places.sqlite
            SQLite version 3.7.13 2012-06-11 02:05:22
            Enter ".help" for instructions. Enter SQL statements terminated with a ";"
Miramos la tablas con   .tables
            moz_anno_attributes  moz_favicons         moz_items_annos
            moz_annos            moz_historyvisits    moz_keywords
            moz_bookmarks        moz_hosts            moz_places
            moz_bookmarks_roots  moz_inputhistory
Miramos la tabla moz_hosts:    SELECT * FROM moz_hosts;
            1|mozilla.org|140|0|
            2|sysdw.nteum.org|11180|1|
            3|127.0.0.1|2000|1|
            4|10.0.0.58|0|0|
Miramos la tabla moz_bookmarks (antes hacemos un bookmark, por ejemplo a SQLite.org: SELECT *
FROM moz_bookmarks;
            ...
            24|1|164|2|4|SQLite Home Page|||1404403846763242|1404403846777354|yfpoHwYrL7Ys
```

En http://www.sqlite.org/quickstart.html tendremos una guía de cómo acceder a través de las diferentes interfaces de programación a la BD y documentación para trabajar con la línea de comandos http://www.sqlite.org/cli.html. La documentación la podemos encontrar en http://www.sqlite.org/docs.html.

1.6. Source Code Control System

Los sistemas de Control de Revisiones (o también conocidos como control de versiones/código fuente) se refieren a la gestión de los cambios en los archivos, programas, páginas o cualquier información que es modificada y actualizada, ya sea un único usuario para llevar un control que ha hecho y cuando o por múltiples usuarios que trabajan simultáneamente en el mismo proyecto. Los cambios se identifican generalmente por un número o código, denominado "número de revisión", "nivel de revisión", "revisión" y el sistema de control permiten volver atrás, aceptar un cambio, publicarlo o saber quién lo ha realizado y qué ha hecho. El sistema trabaja con marcas de tiempos, repositorios

(para guardar las diferentes versiones) y usuarios y permitirá, en función de estos parámetros, comparar los repositorios/archivos, restaurarlos y/o fusionarlos. En la actualidad es una técnica aplicada en grandes aplicaciones informáticas (como por ejemplo, en Google Docs o Wikipedia -o en cualquier wiki-) pero existen diferentes aplicaciones que pueden trabajar como servidores o aplicaciones internas o en modo cliente servidor en función de las necesidades del proyecto/usuarios como por ejemplo CVS (unos de los iniciales junto con RCS), subversion (muy extendido), Git (muy utilizado en proyectos Open Source) o Mercurial (muy utilizado por su aspecto de distribuido -no es necesario un servidor-).

1.6.1. Concurrent Versions System (CVS)

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 controla. 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. [13].

En primer lugar, se debe instalar el Concurrent Versions System (CVS) desde la distribución 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
dfmkdir $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 import
/home/nteum/dir_org/: a.c b.c c.c Makefile              Después del 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 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 [13]. Como interfaces gráficas pueden consultar tkcvs* [34] desarrollada en Tcl/Tk y que soporta Subversion o la también muy popular, Cervisia [35] (ambas en Debian).

***http://www.twobarleycorns.net/tkcvs.html**

1.6.2. 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 subversion-tools`, si deseamos publicar los repositorios en Apache2 deberemos tener instalado este, además del módulo específico `apt-get install libapache2-svn`. Además de paquete Subversion, existen en los repositorios (de la mayoría de distribuciones) paquetes complementarios, como por ejemplo, subversion-tools (herramientas complementarias), svn-load (versión mejorada para la importación de repositorios), svn-workbench (*Workbench* para Subversion), svnmailer (herramienta que extiende las posibilidades de las notificaciones del *commit*).

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
```

```
adduser svuser
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 `id`).

```
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
```

Considerando que el repositorio lo tenemos en /usr/local/svn/pruebas/ a continuación, creamos un directorio temporal en nuestro directorio:

```
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:///usr/local/svn/pruebas/ -m "Ver. Inicial"
```

Si lo hemos creado en `$HOME/svndir/pruebas`, deberíamos reemplazar por *file:///$HOME/svndir/pruebas*. 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/svn-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/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 `svn 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 decir, 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 (`a2enmod dav dav_fs`) e instalar la librearía como se especificó antes `apt-get install libapache2-svn` verificando que el módulo dav_svn esté habilitado también (lo hará la instalación de la librería). A continuación creamos un archivo en */etc/apache2/conf.d/svn.conf* con el siguiente contenido:

Enlace de interés

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

```
<location "/svn">
DAV svn
SVNParentPath /usr/local/svn
SVNListParentPath On
   AuthType Basic
AuthName "Subversion"
AuthUserFile /etc/apache2/htpasswd
Require valid-user
</location>
```

Donde debemos apuntar hacia donde se encuentra nuestro repositorio con validación de usuario (debe ser un usuario válido creado con la instrucción `htpasswd /etc/apache2/htpasswd user`), reiniciamos Apache2 y ya tendremos este habilitado y disponible para acceder a través de la dirección URL: http://sysdw.nteum.org/svn/. Otra forma de acceder es a través de un cliente en forma local o en forma remota que es una opción útil para repositorios pequeños y medianos. Debian incluye una serie de ellos (algunos compatibles con CVS) como por ejemplo **rapidsvn**, **subcommander**, **svnkit** (en java), **tkcvs**, **viewvc** (estos dos últimos soportan también repositorios CVS) y **websvn** (en PHP). Si se desea trabajar en remoto se deberá poner en marcha el servidor de svn con `svnserve -d` y cambiar el archivo *svnserve.conf* para permitir el acceso y el modo (p.ej., quitando el comentario a la línea `anon-access = read`). Este archivo está en el directorio */Dir_Repo/conf* que es donde hemos inicializado el repositorio (en nuestro caso se encuentra en */usr/local/svn/pruebas/conf/svnserve.conf*).

Si deseamos trabajar con un repositorio gestionado via URL lo más práctico es crear uno y gestionarlo desde la consola para insertar los archivos y via web

para ponerlos a disposición del público. En este caso deberemos hacer algunas modificaciones con los permisos para que podamos trabajar con URLs. Crearemos un segundo repositorio para este ejemplo pero haremos que el usuario **svuser** genere sus copias dentro del repositorio:

Como root, hacemos:

```
  mkdir /var/svn Nuestro nuevo repositorio
  svnadmin create /var/svn Creo el repositorio
  chown -R www-data:www-data Para que Apache pueda acceder al directorio
  ls -s /var/svn
-rw-r-r-   1 www-data www-data   229 Jul 4 20:27 README.txt
drwxr-xr-x   2 www-data www-data   4096 Jul 4 20:27 conf
drwxr-xr-x   2 www-data www-data   4096 Jul 4 20:27 dav
drwxr-xr-x   2 www-data www-data   4096 Jul 4 20:28 db
-rw-r-r-   1 www-data www-data   2 Jul 4 20:27 format
drwxr-xr-x   2 www-data www-data   4096 Jul 4 20:27 hooks
drwxr-xr-x   2 www-data www-data   4096 Jul 4 20:27 locks
```

Para la autentificación, se utiliza `htpasswd` (por ejemplo, con la instrucción `htpasswd -c -m /etc/apache2/htpasswd user` donde el parámetro *-c* solo se debe poner la primera vez cuando se debe crear el archivo. A continuación, creamos un archivo en */etc/apache2/conf.d/svn2.conf* por algo como:

```
<location /svn2>
   DAV svn
   SVNPath /var/svn
   AuthType Basic
   AuthName "Subversion Repository"
   AuthUserFile /etc/apache2/htpasswd
   Require valid-user
</location>
```

Reiniciamos Apache y ya estamos listos para importar algunos archivos, por ejemplo desde la consola como svuser:

```
  svn import file1.txt http://sysdw.nteum.org/svn/file1.txt -m "Import
Inicial"
```

Nos pedirá la autentificación y nos dirá que el fichero `file1.txt` se ha añadido al repositorio. Desde la URL http://sysdw.nteum.org/svn2/ veremos el file1.txt.

1.6.3. 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 aplicaciones 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 de Git, Gitolite y Gitweb

Instalaremos un servidor de control de versiones Git sobre Debian Wheezy para que pueda gestionar los archivos de código fuente de un equipo de desarrollo. Para ello, utilizaremos Gitolite (como método de control de acceso de nuestros programadores a los repositorios de software), que es una capa de autentificación a Git basado en ssh y que permite especificar permisos no solo por repositorio, sino también por rama o etiquetas dentro de cada repositorio. Finalmente, instalaremos Gitweb para el acceso a los repositorios y adaptaremos una interfaz basada en http://kogakure.github.io/gitweb-theme/. [17, 18, 19]

Sobre el servidor instalamos los paquetes:

```
apt-get install git-core git-doc gitolite git-daemon-run gitweb
highlight
```

Creamos un usuario para el servicio git: `adduser git`

Adoptamos su identidad y configuramos:

```
su - git
git config --global user.name "git"
git config --global user.email git@sysdw.nteum.org
```

Se puede verificar con: `git config -l`

Se deben generar las llaves pública de los usuarios y enviar al servidor, por lo cual podemos hacer como usuario adminp:

```
ssh-keygen
scp .ssh/id_rsa.pub git@sysdw.nteum.org:/tmp/adminp.pub
```

Si el usuario adminp está en el mismo servidor utilizar `cp` en lugar de `scp`, pero se supone que es otra máquina.

En el servidor activamos la llave de adminp (y la de todos los usuarios), como usuario git hacemos `gl-setup /tmp/adminp.pub` Esto abrirá un editor (de archivo */home/git/.gitolite.rc*) con indicaciones en el cual deberemos cambiar la línea $REPO_UMASK = 0027; (en lugar de 77 cambiar por 27).

A continuación cambiar los permsisos:

```
chmod g+r /home/git/projects.list
chmod -R g+rx /home/git/repositories
```

Editar */etc/group* y agregar el usuario www-data al grupo git modificando la línea a: git:x:1001:www-data

En la máquina local y como usuario adminp (también se puede hacer en la misma máquina del servidor en la cuenta del usuario adminp) Cargaremos la configuración de gitolite y agregaremos dos repositorios:

```
git clone git@sysdw.nteum.org:gitolite-admin.git
cd gitolite-admin
```

Editamos el archivo *conf/gitolite.conf* con los repositorios:

```
repo    gitolite-admin
RW+     =   adminp

repo    testing
    RW+     =   @all

repo wiki
    RW+ = @all
    R = daemon
    R = gitweb

repo data
    RW+ = adminp remix
    R = daemon
    R = gitweb
```

Luego agregamos al repositorio: `git add *`

Y hacemos el commit: `git commit -m "Repositorios wiki y data con permiso de escritura para admin y remix"`

Lo subimos: `git push origin master`

Para la configuración de gitweb editamos */etc/gitweb.conf* modificando las líneas indicadas:

```
$projectroot = "/home/git/repositories";
$projects_list = "/home/git/projects.list";
```

Modificamos */etc/sv/git-daemon/run* (antes hacer una copia como run.org por ejemplo)

```
#!/bin/sh
  exec 2>&1
  echo 'git-daemon starting.'
  exec chpst -ugitdaemon:git \
       "$(git -exec-path)" /git-daemon -verbose -reuseaddr \
       -base-path=/home/git/repositories /home/git
```

Luego reiniciamos el daemon: `sv restart git-daemon`

También reiniciamos Apache2 y en la URL http://sysdw.nteum.org/gitweb/ tendremos los proyectos.

Para cambiar el aspecto de la página web con http://kogakure.github.io/gitweb-theme/. Podemos descargar el archivo tgz de esta página, lo descomprimimos y desde dentro del directorio copiamos el contenido

```
cp * /usr/share/gitweb/static/
```

Si recargamos el directorio borrando la caché (Crtl+F5) en el navegador deberíamos ver el nuevo aspecto. Un aspecto interesante es ver ahora cómo desde el usuario admin insertamos archivos en uno de los repositorios y los publicamos:

Creo un directorio *hello* con dos archivos uno archivo llamado *hello.c* y otro *library.h*

```
/* Hello.c */
#include <stdio.h>
#include "library.h"
    int main (void) {printf ("Hola UOC!\n");}
/* library.h */
#ifndef DEFINITIONS_H
#define DEFINITIONS_H 1
#endif /* DEFINITIONS_H */
```

Dentro del directorio ejecuto:

```
git init
git add .
git commit -m "initial commit"
git remote add origin git@sysdw.nteum.org:wiki.git
git push origin master
```

Actualizando la página web veremos que ya tenemos los archivos en el repositorio Si quisiéramos clonar el repositorio solo deberíamos hacer:

```
git clone git@sysdw.nteum.org:wiki.git
```

Y donde estamos creará un directorio wiki con los archivos hello.c library.h (además de otro .git que es donde se encuentra toda la información de Git).

1.6.4. Mercurial

Mercurial es un sistema de control de versiones (escrito en su mayor parte en python y algunas partes en C -diff-) diseñado para facilitar la gestión de archivos/directorios a los desarrolladores de software. El uso básico de Mercurial es en línea de comandos y existe un comando que es el que gestiona todo el paquete a través de opciones llamado `hg` (en referencia al símbolo químico del mercurio). Las premisas de diseño de mercurial son el rendimiento rendimiento y la escalabilidad a través de un desarrollo distribuido y sin necesidad de un servidor. Además presenta una gestión robusta de archivos tanto de texto como binarios y posee capacidades avanzadas de ramificación e integración y una interfaz web integradas.[21]

Su instalación es muy simple con `apt-get mercurial` y es interesante instalar otro paquete llamado Tortoisehg [23] (`apt-get install tortoisehg`), que es una extensión gráfica a Mercurial y actúa como *Workbench*, soportando múltiples repositorios, y permite resolver conflictos en una forma simple y gráfica. Después de la instalación podremos ejecutar el comando `hg version` y `hg debuginstall` donde este último nos dirá que debemos identificar al usuario e incluir una dirección de mail. Para ello, creamos el archivo more $HOME/.hgrc con el siguiente contenido (la segunda parte es para Tortoise):

```
[ui]
username = AdminP <adminp@sysdw.nteum.org>
[extensions]
graphlog =
```

Con ello ya estamos en condiciones de comenzar a trabajar en nuestro repositorio. Para usuario recién iniciados es muy recomendable seguir la guía

de [22] que muestra el procedimiento para gestionar un repositorio entre dos usuarios. Una secuencia de comandos podría ser:

Creo un repositorio: `hg init hello` veremos un directorio con un subdirectorio *.hg*
Dentro de *hello* creo un archivo: `echo "Hello World" > hello.txt`
Miro el estado pero veremos que no forma parte de repositorio: `hg status`
Lo agrego y si ejecutamos la orden anterior veremos que ya está dentro: `hg add hello.txt`
Para publicarlo en el repositorio: `hg commit -m "Mi primer hg-hello"`
La historia la podemos ver con: `hg log`
Modifico y miro las diferencias: `echo "Bye..." >hello.txt; hg diff`
Puedo revertir los cambios: `hg revert hello.txt`
Añado información: `echo "by Admin." >> hello.txt`
Publico los cambios: `hg commit -m "Added author."`
Pregunto a HG donde están estos: `hg annotate hello.txt`
Las anotaciones son de mucha ayuda para fijar los errores ya que nos permite ver que y quién cambió:
`hg log -r 1` y en forma espacial `hg glog` o mejor en forma gráfica utilizando Tortoise `hg glog`
Que nos mostrará que el cambio es hijo de la misma rama-autor.
Miro los parientes y vuelvo atrás (a una versión anterior): `hg parents` luego `hg update 0` si hacemos `hg parents` y miramos el archivo veremos que tenemos la versión original.
Pero no lo hemos perdido, si queremos mirar lo anterior: `hg cat -r 1 hello.txt`
Y recuperarlo: `hg update`
Si queremos trabajar entre dos usuarios que comparten sistema de archivos (podría ser por SSH o HTTP también), el usuario interesado podrá hacer una copia del repositorio: `hg clone ../adminp/hello`
Incluso se podría hacer por mail/USB utilizando `hg bundle` para generar un binario y llevárselo a otra máquina. Este nuevo usuario será independiente de lo que haga el otro usuario (adminp)
Este usuario (remix) podrá hacer:
`cd hello; echo "My hola" > hello2.txt; echo "by Remix." >> hello2.txt`
Lo agrega a su repositorio y publica: `hg add` y `hg commit -m "My hello."`
Luego modifico lo de Adminp: `echo "not by Remix" >> hello.txt`
Publico: `hg commit -m "Not my file."`
Y puedo preguntar si han habido cambios con respecto a adminp clone: `hg incoming`
Y preguntar que cambios no están en adminp: `hg outgoing`
Remix no tiene acceso al directorio de Adminp por lo cual no puede introducir los cambios pero Adminp si puede leer y agregarlos. Debe poner en su archivo *.hg/hgrc*

```
[paths]
default = /home/remix/hola
```

Y adminp puede observar los cambios con *hg incoming*
Y agregarlos con: *hg pull* es interesante verlos con Tortoise que lo mostrará gráficamente.
Es interesante que de una forma simple este repositorio lo podemos publicar via web haciendo en nuestro repositorio: `hg serve`
A partir de ello se podrá acceder en la URL http://sysdw.nteum.org:8000 (o en localhost)
Esta solución es simple (pero eficiente) y temporal. Repositorios más estables deberán ser publicados utilizando Apache con el script *hgweb.cgi/hgwebdir.cgi* indicado en el manual de instalación. [25][26]
También es interesante el comando `hg-ssh` cuando los repositorios se ofrecen a través de SSH.

Es interesante completar la información con un tutorial [24] y ver las extensiones que admite HG [28] y no menos interesantes son los paquetes (incluidos en Debian) **hgview** (*interactive history viewer*), **mercurial-git** (git *plugin* para mercurial), **mercurial-server** (*shared Mercurial repository service*), o **eclipse-mercurialeclipse** (integración con Eclipse).

1.7. Mantis Bug Tracker

Mantis Bug Tracker es un sistema (GPL) 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. Además del paquete de Mantis, es necesario tener instalado MySQL, Apache Web Server y el módulo PHP para Apache. La instalación de Mantis es muy simple `apt-get install mantis` y durante el proceso nos indicará que la contraseña para el usuario administrator (y el aviso de que es recomendable cambiarla) y pedirá alguna información más en función de los paquetes que tengamos instalados (Apache, MySQL, MariaDB, ...). 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 (*ítem manage*), 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.. Se recomienda por crear nuevos usuarios, nuevas categorías y luego proyectos asignadas a estas categorías. A partir de ese momento se puede insertar incidencias, asignándolas y gestionándolas a través de la interfaz. Si tenemos instalada MariaDB nos podrá dar un error similar a *mysql_connect(): Headers and client library minor version mismatch* y deberemos actualizar los *headers y drivers* de PHP `apt-get install php5-mysqlnd`.

Hay un amplio conjunto de opciones que se pueden modificar en el archivo */usr/share/mantis/www/config_inc.php* [30], 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 blanco 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
```

***http://www.mantisbt.org /wiki/doku.php /mantisbt:mantis_recipies Para más configuraciones o integraciones con otros paquetes, consultad el manual [29] o acceded a la página de la *wiki* del proyecto* [30].**

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 o con MariaDB.

3. Utilizando las tablas de World http://pgfoundry.org/projects/dbsamples/ obtened la población promedio de las poblaciones de las ciudades entre 10.000 y 30.000 habitantes, y un porcentaje de cuál es la lengua más hablada y la menos hablada con relación a los habitantes.

4. 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.

5. 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.

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

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

8. 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.

9. Repetid el paso anterior con Mercurial.

10. Instalad Mantis, generad 3 usuarios, 2 categorías, 2 proyectos y 3 incidencias para cada usuarios, gestionándolas a través de la interfaz. Generad un usuario anónimo que pueda acceder a un proyecto público como reporter, pero no a uno privado.

Bibliografía

[1] *Big Data - Infografía*.
<http://i2.wp.com/unadocenade.com/wp-content/uploads/2013/05/Infograf%C3%ADa-Big-Data.jpg>

[2] **M. Gibert, O. Pérez**. *Bases de datos en PostgreSQL*.
<http://ocw.uoc.edu/computer-science-technology-and-multimedia/bases-de-datos/bases-de-datos/P06_M2109_02152.pdf>

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

[4] **PostgreSQL**.*The SQL Language*.
<http://www.postgresql.org/docs/9.1/interactive/tutorial-sql.html>

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

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

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

[8] *Mysql Administrator*.
<http://www.mysql.com/products/workbench/>

[9] *MariaDB*.
<https://mariadb.org/>

[10] *Setting up MariaDB Repositories*.
<https://downloads.mariadb.org/mariadb/repositories/>

[11] *SQLite*.
<http://www.sqlite.org/>

[12] *SQLite Database Browser*.
<https://github.com/sqlitebrowser/sqlitebrowser>

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

[14] *Libro sobre Subversion*.
<http://svnbook.red-bean.com/nightly/es/index.html>
(versión en español)

[15] *Subversion Documentation*.
<http://subversion.tigris.org/servlets/ProjectDocumentList>)

bibitemsub *Subversion*.
<http://subversion.apache.org/>

[16] *Multiples repositorios con Subversion*.
<http://www.debian-administration.org/articles/208>

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

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

[19] **L. Hernández**.*Servidor Git + Gitolite + Gitweb sobre Debian*.
<http://leninmhs.wordpress.com/2014/01/19/git-gitolite-gitweb/>

[20] **D. Mühlbachler**.*How-To: Install a private Debian Git server using gitolite and GitLabHQ*.
<http://blog.muehlbachler.org/2012/01/how-to-install-a-private-debian-git-server-using-gitolite-and-gitlabhq/>

[21] *Mercurial*.
<http://mercurial.selenic.com/>

[22] *Basic Mercurial*.
<http://mercurial.aragost.com/kick-start/en/basic/>

[23] *TortoiseHg Docs*.
<http://tortoisehg.bitbucket.org/docs.html>

[24] *HgInit: a Mercurial tutorial*.
<http://hginit.com/>

[25] *Publishing Repositories with hgwebdir.cgi*.
<http://mercurial.selenic.com/wiki/HgWebDirStepByStep>

[26] *Publishing Mercurial Repositories*.
<http://mercurial.selenic.com/wiki/PublishingRepositories>

[27] *Remote Repositories*.
<http://mercurial.aragost.com/kick-start/en/remote/>

[28] *Using Mercurial Extensions*.
<http://mercurial.selenic.com/wiki/UsingExtensions>

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

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

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

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

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

[34] *TkCVS: Tcl/Tk-based graphical interface to the CVS*.
<http://www.twobarleycorns.net/tkcvs.html>

[35] *Cervisia - CVS Frontend*.
<http://www.kde.org/applications/development/cervisia/>