Data administration

Remo Suppi Boldrito

PID_00148469

Universitat Oberta de Catalunya
www.uoc.edu

Index

Introduction 5

1. PostgreSQL 7
1.1. How should we create a DB? 7
1.2. How can we access a DB? 8
1.3. SQL language 8
1.4. Installing PostgreSQL 10
1.4.1. Post-installation 11
1.4.2. DB users 12
1.5. Maintenance 14
1.6. Pgaccess 15

2. Mysql 17
2.1. Installation 17
2.2. Post-installation and verification 18
2.3. The MySQL monitor program (client) 19
2.4. Administration 21
2.5. Graphic interfaces 22

3. Source Code management systems 24
3.1. Revision control system (RCS) 25
3.2. Concurrent versions system (CVS) 25
3.2.1. Example of a session 29
3.2.2. Multiple users 30
3.3. Graphic interfaces 30

4. Subversion 32

Activities 37

Bibliography 38

Introduction

An important aspect of an operating system is where and how the data is saved. When the availability of data needs to be efficient, it is necessary to use databases (DB).

A database is a structured set of data that can be organised in a simple and efficient manner by the database handler. Current databases are known as relational, since the data can be stored in different tables for ease of management and administration. For this purpose and with a view to standardising database access, a language known as structured query language (SQL) is used. This language allows a flexible and rapid interaction irrespective of the database applications.

At present, the most commonly used way consists of accessing a database from an application that runs SQL code. For example, it is very common to access a DB through a web page that contains PHP or Perl code (the most common ones). When a page is requested by a client, the PHP or Perl code embedded in the page is executed, the DB is accessed and the page is generated with its static content and the content extracted from the DB that is then sent to the client. Two of the most relevant current examples of databases are those provided by ProstgreSQL and MySQL, which are the ones we will analyse.

However, when we work on software development, there are other data-related aspects to consider, regarding their validity and environment (especially if there is a group of users that work on the same data). There are several packages for version control (revisions), but the purpose of all of them is to facilitate the administration of the different versions of each developed product together with the potential specialisations made for any particular client.

Versions control is provided to control the different versions of the source code. However, the same concepts apply to other spheres and not only for source code but also for documents, images etc. Although a version control system can be implemented manually, it is highly advisable to have tools that facilitate this management (cvs, Subversion, SourceSafe, Clear Case, Darcs, Plastic SCM, RCS etc.).

In this chapter, we will describe cvs (*version control system*) and Subversion for controlling and administering multiple file revisions, automating storage, reading, identifying and merging different revisions. These programs are useful when a text is revised frequently and includes source code, executables, libraries, documentation, graphs, articles and other files. [Pos03e, Mys, Ced]

The reasoning behind using cvs and Subversion is that cvs is one of the most commonly used traditional packages and Subversion is a version control system software designed specifically to replace the popular cvs and to resolve several of its deficiencies. Subversion is also known as svn since this is the name of the command line tool. An important feature of Subversion is that, unlike CVS, the files with versions do not each have an independent revision number. Instead, the entire repository has a single version number that identifies a shared status of all the repository's files at a certain point in time.

1. PostgreSQL

The PostgreSQL database (DB) language uses a client server model [Posa]. A PostgreSQL session consists of a series of programs that cooperate:

- A server process that handles the DB files accepts connections from clients and performs the actions required by the clients on the DB. The server program in PostgreSQL is called postmaster.

- The client application (frontend) is what requests the operations to be performed on the DB, which can be extremely varied; for example: tools in text mode, graphic, web servers etc.

Generally, the client and the server are on different hosts and communicate through a TCP/IP connection. The server can accept multiple requests from different clients, activating a process that will attend to the user's request exclusively and transparently for each new connection. There is a set of tasks that can be performed by the user or by the administrator, as appropriate, and that we describe as follows.

1.1. How should we create a DB?

The first action for checking whether the DB server can be accessed is to create a database. The PostgreSQL server can handle many DBs and it is recommended to use a different one for each project. To create a database, we use the createdb command from the operating system's command line. This command will generate a *CREATE DATABASE* message if everything is correct. It is important to take into account that for this action we will need to have a user enabled to create a database. In the section on installation (1.4) we will see that there is a user, the one that installs the database, who will have permissions for creating databases and creating new users who in turn can create databases. Generally (and in Debian) the default user is postgres. Therefore, before running createdb, we need to run postgres (if we are the root user, we do not need a password, but any other user will need the postgres password) and then we will be able to run createdb. To create a DB named nteumdb:

```
createdb nteumdb
```

If we cannot find the command, it may be that the path is not properly configured or that the DB is not properly installed. We can try with the full path (`/usr/local/pgsql/bin/createdb nteumdb`), which will depend on our specific installation, or check references for problem-solving. Other messages

would be *could not connect to server* when the server has not initiated or *CREATE DATABASE: permission denied* when we do not have authorisation to create the DB. To eliminate the DB, we can use `dropdb nteumdb`.

1.2. How can we access a DB?

After we have created the DB, we can access it in various ways:

- By running an interactive command called psql, which allows us to edit and execute SQL commands (e.g. `psql nteumdb`).
- Executing a graphic interface such as PgAccess or a suite with ODBC support for creating and manipulating DBs.
- Writing an application using one of the supported languages, for example PHP, Perl, Java... (see PostgreSQL 7.3 Programmer's Guide).

See also

In order to access the DB, the database server must be running. When we install Postgresql the appropriate links are created so that the server initiates when the computer boots. For more details, consult the section on installation (1.4).

For reasons of simplicity, we will use psql to access the DB, meaning that we will have to enter `psql nteumdb`: some messages will appear with the version and information and a similar prompt to nteumdb =>. We can run some of the following SQL commands:

```
SELECT version();
```

or also

```
SELECT current date;
```

psql also has commands that are not SQL and that start with '\', for example \h (list of all available commands) or \q to finish.

Example

```
Access the DB nteumdb:
psql nteumdb [enter]
nteumdb =>
```

1.3. SQL language

The purpose of this section is not to provide a tutorial on SQL, but we will analyse some examples to see this language's capabilities. They are examples that come with the PostgreSQL distribution in the InstallationDirectory/src/tutorial directory; in order to access them, change to the PostgreSQL directory (`cd InstallationDirectory/ src/tutorial`) and run `psql -s nteumdb` and once inside \i basics.sql. The parameter \i reads the commands of the specified file (basic.sql in this case).

PostgreSQL is a relational database management system (RDBMS), which means that it manages data stored in tables. Each table has a specific number of rows and columns and every column contains a specific type of data. The tables are grouped into one DB and a single server handles this collection of DB (the full set is called a database cluster).

To create, for example, a table with psql, run:

```
CREATE TABLE weather (
      city               varchar(80),
      min_temp           int,
      max_temp           int,
      real               rain,
      day                date
);
```

Example

Create table. Inside *psql:*

CREATE TABLE NameTB (var1 type, var2 type,...);

The command ends when we type ';' and we can use blank spaces and tabs freely. varchar(80) specifies a data structure that can store up to 80 characters (in this case). The point is a specific type of PostgreSQL.

To delete the table:

DROP TABLE *table_name ;*

We can enter data in two ways, the first is to enter all the table's data and the second is to specify the variables and values that we wish to modify:

INSERT INTO weather VALUES ('Barcelona', 16, 37, 0.25, '2007-03-19');
INSERT INTO weather (city, min_temp, max_temp, rain, day)

VALUES ('Barcelona', 16, 37, 0.25, '2007-03-19'); This method can be simple for small amounts of data, but when a large amount of data has to be entered, it can be copied from a file with the sentence:

Example

A second example could be:

```
CREATE TABLE city(
name varchar(80),
      place
      point
);
```

COPY weather FROM '/home/user/time.txt'; (this file must be on the server, not on the client).

To look at a table, we could type:

SELECT * FROM weather;

where * means all the columns.

Recommend Reading

We recommend studying chapter 3 of PostgreSQL on advanced characteristics (Views,Foreign Keys, Transactions, http://www.postgresql.org/docs/8.2/static/tutorial-advanced.html [Pos03d]

Examples

- Enter the data into the table. Inside psql:

```
INSERT INTO TBName (valueVar1, ValueVar2,...);
```

- Data from a file. Inside psql:

```
COPY TBName FROM 'FileName';
```

- Visualising data. Inside *psql*:

```
SELECT * FROM TBName;
```

Examples of more complex commands (within *psql*) would be:

- Visualises the column city after typing:

```
SELECT city, (max_temp+min_temp)/2 AS average_temp, date FROM weather;
```

- Visualises everything where the logical operation is fulfilled:

```
SELECT * FROM weather WHERE city = 'Barcelona'
    AND rain \verb+>+ 0.0;
```

- Joining tables:

```
SELECT * FROM weather, city WHERE city = name;
```

- Functions, in this case maximum:

```
SELECT max(min_temp) FROM weather;
```

- Nested functions:

```
SELECT city FROM weather WHERE min_temp = (SELECT max(min_temp) FROM weather);
```

- Selective modification:

```
UPDATE weather SET max_temp = max_temp 2, min_temp = min_temp 2 WHERE day > '19990128';
```

- Deleting the register:

```
DELETE FROM weather WHERE city = 'Sabadell';
```

1.4. Installing PostgreSQL

This step is necessary for DB administrators [Posa]. The DB administrator's functions include software installation, initialisation and configuration, administration of users, DBs and DB maintenance tasks.

The database can be installed in two ways: through the distribution's binaries, which is not difficult, since the distribution scripts carry out all the necessary steps for making the DB operative, or through the source code, which will have to be compiled and installed. In the first case, we can use the *kpackage* (Debian) or the *apt-get*. In the second case, we recommend always going to the source (or to a mirror repository of the original distribution). It is important to bear in mind that the installation from the source code will then be left outside the DB of installed software and that the benefits of software administration offered, for example, by *apt-cache* or *apt-get* will be lost.

Installation from source code step by step:

- First we need to obtain the software from the site (x.x is the available version) http://www.postgresql.org/download/ and decompress it (x.x.x is version number 8.2.3 at the time of this revision):
  `gunzip postgresql-x.x.x.tar.gz`
  `tar xf postgresql-7.3.tar`

- Change to the postgresql directory and configure it with `./configure`.

- Compile it with gmake, verify the compilation with `gmake check` and install it with `gmake install` (by default, it will install it in /usr/local/pgsql).

1.4.1. Post-installation

Initialise the variables, in *bash, sh, ksh*:

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

or, in csh:

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

We recommend locating this initialisation in the user configuration scripts, for example /etc/profile or .bashrc for *bash*. To have access to the manuals, we need to initialise the MANPATH variable in the same way:

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

Once the DB is installed, we will need to create a user that will handle the databases (it is advisable to create a different user from the root user so that there is no connection with other services of the machine), for example, the postgres user using the *useradd*, command for example.

Next, we will need to create a storage area for the databases (single space) on the disk, which will be a directory, for example /usr/local/pgsql/data. For this purpose, execute `initdb -D /usr/local/pgsql/data`, connected as the user created in the preceding point. We may receive a message that the directory cannot be created due to no privileges, meaning that we will first have to create the directory and then tell the DB which it is; as root, we have to type, for example:

```
mkdir /usr/local/pgsql/data
```

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

Initiate the server (which is called postmaster), to do so, use:

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

to run it in active mode (in the foreground); and to run it in passive mode (in the background) use:

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

Reroutings are done in order to store the server's errors. The package also includes a script (pg_ctl) so as not to have to know all the *postmaster* syntax in order to run it:

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

We can abort the server's execution in different ways, with pg-ctl, for example, or directly using:

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

1.4.2. DB users

DB users are completely different to the users of the operating system. In some cases, it could be interesting for them to maintain correspondence, but it is not necessary. The users are for all the DBs that the server controls, not for each DB. To create a user, execute the SQL sentence:

Note

Creating, deleting users:
createuser [options] name
dropuser [options] name

CREATE USER *name*

To delete users:

DROP USER *name*

We can also call on the createuser and dropuser programs from the command line. There is a default user called *postgres* (within the DB), which is what will allow us to create the rest (to create new users from `psql -U postgres` if the user of the operating system used for administrating the DB is not *postgres*).

A DB user can have a set of attributes according to what the user is allowed to do:

- Superuser: this user has no restrictions. For example, it can create new users; to do this, run:
  CREATE USER name CREATEUSER

- DB creator: is authorised to create a DB. To create a user with these characteristics, use the command:
  CREATE USER name CREATEDB

- Password: only necessary if we wish to control users' access when they connect to a DB for security reasons. To create a user with a password, we can use:
  CREATE USER name PASSWORD 'password'
  where password will be the password for that user.

- We can change a user's attributes by using the command ALTER USER. We can also make user groups that share the same privileges with:
  CREATE GROUP GroupName
  And to insert the users in this group:
  ALTER GROUP GroupName ADD USER Name1
  Or to delete it :
  ALTER GROUP GroupName DROP USER Name1

Example

Group operations inside psql:

CREATE GROUP *GroupName;*
ALTER GROUP *GroupName* ADD USER *Name1...*; ALTER GROUP *GroupName*
DROP USER *Name1...*;

When we create a DB, the privileges are for the user that creates it (and for the superuser). To allow another user to use this DB or part of it, we need to grant it privileges. There are different types of privileges such as SELECT, INSERT, UPDATE, DELETE, RULE, REFERENCES, TRIGGER, CREATE, TEMPORARY, EXECUTE, USAGE, and ALL PRIVILEGES (consult the references for their meaning). To assign privileges, we can use:

GRANT UPDATE ON object TO user

where user must be a valid PostgreSQL user and object, a table, for example. This command must be executed by the superuser or table owner. The PUBLIC user can be used as a synonym for all users and ALL, as a synonym for all privileges. For example, to remove all of the privileges from all of the users of an object, we can execute:

REVOKE ALL ON object FROM PUBLIC;

1.5. Maintenance

There is a set of tasks that the DB administrator is responsible for and that must be performed periodically:

1) Recovering the space: periodically we must execute the VACUUM command, which will recover the disk space of deleted or modified rows, update the statistics used by the PostgreSQL scheduler and improve access conditions.

2) Reindexing: In some cases, PostgreSQL can give problems with the reuse of indexes, therefore it is advisable to use REINDEX periodically to eliminate pages and rows. We can also use contrib/reindexdb in order to reindex an entire DB (we need to take into account that, depending on the size of the DBs, these commands can take a while).

3) Change of log files: we need to prevent the log files from becoming too large and difficult to handle. This can be done easily when the server is initiated with:

```
pg_ctl start | logrotate
```

logrotate renames and opens a new log file and it can be configured with /etc/logrotate.conf.

4) Backup copy and recovery: there are two ways of saving data, with the sentence SQL Dump or by saving the DB file. The first will be:

```
pg_dump DBFile> BackupFile
```

For recovery, we can use: `psql DBFile< BackupFile`

In order to save all of the server's DBs, we can execute:

```
pg_dumpall > TotalBackupFile
```

Another strategy is to save the database files at the level of the operating system, for example using:

```
tar -cf backup.tar /usr/local/pgsql/data
```

There are two restrictions that can make this method unpractical:

- The server has to be stopped before saving and recovering the data.

- We need to know very well all of the implications at the level of the file where all the tables are, transactions etc., since otherwise, we could render a DB useless. Also (in general), the size that will be saved will be greater

than if done using the previous methods, since for example, with the pg_dumb the indexes are not saved, but rather the command in order to recreate them is saved.

Summary of the installation of PostgreSQL:

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

1.6. Pgaccess

The application `pgaccess [DBName]` (http://www.pgaccess.org/) allows us to access and administer a database with a graphic interface. The easiest way of accessing (from KDE for example) is from a terminal, the DB administrator will have to do, (if not the *postgresql* user) `xhost+` which will allow other applications to connect to the current user's display

```
su postgres
pgaccess [DBName]&
```

If configured in 'Preferences' it will always open the last DB. Figure 15 shows the pgaccess interface.

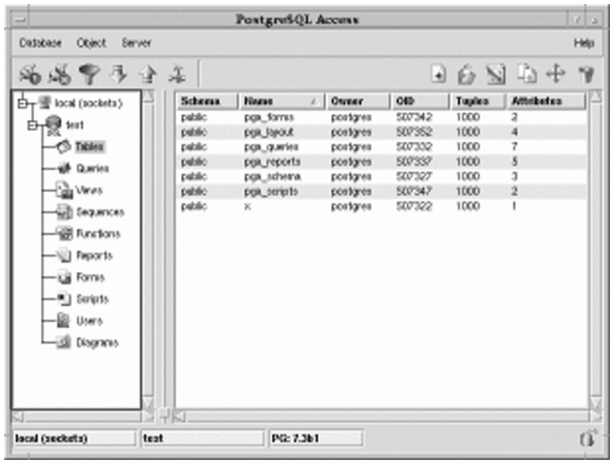

Figure 1. PgAccess

In a typical session the administrator /user could, firstly, Open DataBase, indicating here for example, Port = 5432, DataBase = nteum (the other parameters are not necessary if the database is local) and then Open. As of this moment, the user will be able to work with the bidimensional space selecting what it wants to do in the *Y* axis (tables, consultations, views etc.) and with that el-

ement highlighted, and selecting one of that type within the window, using the *X* axis above for New (add), Open or Design. For example, if we select in *Y* Users and in *X*, New, the application will ask for the username, password (with verification), timeout and characteristics (for example, Create DB, Create other users). In DataBase we could also select Preferences, so as to change the type of font, for example, and select the possibility of seeing the system's tables.

Users' personal configurations will be registered in the file ~/.pgaccessrc. The interface helps to perform/facilitate a large amount of the user/administrator's work and it is recommended for users who have just started in PostgreSQL, since they will not need to know the syntax of the command line as in psql (the application itself will request all of a command's options through several windows).

A simpler tool is through the corresponding webmin module (we need to install the packages webmin-core and required modules, for example, in this case webmin-postgresql), but in many distributions it is not included by default (for more information visit http://www.webmin.com/). During the installation, webmin will warn that the main user will be the root and will use the same password as the root of the operating system. To connect, we can do so from a navigator for example, https://localhost:10000, which will ask to accept (or deny) the use of the SSL certificate for the SSL communication, and next it will show all of the services that can be administered, among them the PostgreSQL Data Base Server.

2. Mysql

MySQL [Mys] is (according to its authors) the most popular open SQL (DB), in other words free software (Open Source), and is developed and distributed by MySQL AB (a commercial enterprise that makes profit from the services it offers over the DB). MySQL is a database management system (DBMS). A DBMS is what can add and process the data stored inside the DB. Like PostgreSQL, MySQL is a relational database, which means that it stores data in tables instead of in a single location, which offers greater speed and flexibility. As it is free software, anyone can obtain the code, study it and modify it according to their requirements, without having to pay anything, since MySQL uses the GPL license. On its webpage, MySQL offers a set of statistics and features compared to other DBs to show users how fast, reliable and easy it is to use. The choice of a DB should be made carefully according to users' needs and the environment in which the DB will be used.

2.1. Installation

- Obtain it from http://www.mysql.com/ or any of the software repositories. The binaries and source files can be obtained for compilation and installation.

- In the case of the binaries, use the Debian distribution, and select the packages mysql-* (client, server, common are required). Following a number of questions, the installation will create a mysql user and an entry in /etc/init.d/mysql in order to start/stop the server during boot. It can also be done manually using:
  `/etc/init.d/mysql start|stop`

- In order to access the database, we can use the mysql monitor from the command line. If we obtain the binaries (not Debian or RPM, with this simply use the common -apt-get, rpm-), for example gz from the MySQL website, we will have to execute the following commands in order to install the 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 &
```

This creates the user/group/directory, decompresses and installs the DB in /usr/local/mysql.

- In the case of obtaining the source code, the steps are similar:

```
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 &
```

It is important to pay attention when configuring, since `prefix= /usr/local/mysql` is the directory where the DB will be installed and it can be changed to locate the DB in any directory we wish.

2.2. Post-installation and verification

Once installed (whether from the binaries or the source code), we will have to verify if the server works properly. In Debian this can be done directly:

`/etc/init.d/mysql start`	starts the server
`mysqladmin version`	Generates version information
`mysqladmin variables`	Shows the values of the variables
`mysqladmin -u root shutdown`	Shuts down the server
`mysqlshow`	Will show the predefined DBs
`mysqlshow mysql`	Will show the tables of the mySQL DB

If installed from the source code, before making these checks we will have to execute the following commands in order to create the databases (from the distribution's directory):

```
./scripts/mysql_install_db
cd InstallationDirectoryMysql
```

```
./bin/mysqld_safe --user = mysql &
```

If we install from the binaries (RPM, Pkg,...), we must do the following:

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

The script `mysql_install_db` creates the `mysql` DB and `mysqld_safe` starts up the mysqld server. Next, we can check all of the commands given above for Debian, except the first one which is the one that starts up the server. Plus, if the tests have been installed, these can be run using `cd sql-bench` and then `run-all-tests`. The results will appear in the directory sql-bech/results for comparison with other DBs.

2.3. The MySQL monitor program (client)

The MySQL client can be used to create and use simple DBs, it is interactive and can connect to the server, run searches and visualise results. It also works in batch mode (as a script) where the commands are passed onto it through a file). To see all the command options, we can run `mysql --help`. We will be able to make a connection (local or remote) using the mysql command, for example, for a connection via the web interface but from the same machine:

Note

MySQL client (frontend):
mysql [DBName]

Web site

For further information, see the documentation, commands and options. [Mys07]
http://dev.mysql.com/doc/refman/5.0/es/

```
mysql -h localhost -u mysql -p DBName
```

If we do not enter the last parameter, no DB is selected.

Once inside, mysql will show a prompt (mysql>) and wait for us to insert a command (own and SQL), for example help. Next, we will give a series of commands in order to test the server (remember always to type the ';' to end the command):

```
mysql> SELECT VERSION(), CURRENT_DATE;
```

We can use capital letters or small caps.

```
mysql> SELECT SIN(PI()/4), (4+1)*5; Calculator.
mysql> SELECT VERSION(); SELECT NOW();
```

Multiple commands on the same line.

```
mysql> SELECT
-> USER()
-> ,
-> CURRENT_DATE;
```

Or on multiple lines.

```
mysql> SHOW DATABASES;
```

Shows the available DBs.

```
mysql> USE test
```

Changes the DB.

```
mysql> CREATE DATABASE nteum; USE nteum;
```

Creates and selects a DB called nteum.

```
mysql> CREATE TABLE pet (name VARCHAR(20), owner VARCHAR(20),
-> species VARCHAR(20), sex CHAR(1), birth DATE, death DATE);
```

Creates a table inside nteum.

```
mysql> SHOW TABLES;
```

Shows the tables.

```
mysql> DESCRIBE pet;
```

Shows the table's definition.

```
mysql> LOAD DATA LOCAL INFILE "pet.txt" INTO TABLE pet;
```

Loads data from pet.txt in pet. The pet.txt file must have one register per line separated by data tabs according to the table's definition (date in YYYY-MM-DD format).

```
mysql> INSERT INTO pet
-> VALUES ('Marciano','Estela','gato','f','1999-03-30',NULL);
```

Loads data inline.

```
mysql> SELECT * FROM pet;Shows table data.
mysql> UPDATE pet SET birth = "1989-08-31" WHERE name = "Browser";
```

Modifies table data.

```
mysql> SELECT * FROM pet WHERE name = "Browser";
```

Selective sample.

```
mysql> SELECT name, birth FROM pet ORDER BY birth;
```

Ordered sample.

```
mysql> SELECT name, birth FROM pet WHERE MONTH(birth) = 5;
```

Selective sample with functions.

```
mysql> GRANT ALL PRIVILEGES ON *.* TO
martian@localhost -> IDENTIFIED BY 'passwd'
WITH GRANT OPTION;
```

Create user marciano in the DB. This has to be executed by the DB root user. Or it can also be done directly by using.

```
mysql> INSERT INTO user (Host,User,Password) ->

VALUES('localhost','marciano','passwd');
```

2.4. Administration

Mysql has a configuration file in /etc/mysql/my.cnf (in Debian), where the DB default options can be changed, for example, the connection port, user, password of remote users, log files, data files, whether it accepts external connections etc. In relation to security, we need to take certain precautions:

1) Not to give anyone (except the root user of MySQL) access to the user table within the MySQL DB, since this is where the user passwords are, which could be used for other purposes.

2) Verify `mysql -u root`. If we can access, it means that the root user does not have a password. To change this, we can type:

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

Now, to connect as root:

```
mysql -u root -p mysql
```

3) Check the documentation concerning the security conditions and the network environment to avoid problems with attacks and/or intrusions.

4) To make copies of the database, we can use the following command:

```
mysqldump --tab = /DestinationDirectory \
--opt DBName
```

or also:

```
mysqlhotcopy DBName /DestinationDirectory
```

Likewise, we can copy the files *.frm', *.MYD', and *.MYI with the server stopped. To recover, execute:

REPAIR TABLE o myisamchk –r

which will work in 99% of cases. Otherwise, we could copy the saved files and start up the server. There are other alternative methods depending on what we want to recover, such as the possibility of saving/recovering part of the DB (see point 4.4 of the documentation). [Mys]

2.5. Graphic interfaces

There are a large number of graphic interfaces for MySQL, among which we should mention MySQL Administrator (it can be obtained from http://www.mysql.com/products/tools/administrator/). Also as tools we can have Mysql-Navigator (http://sourceforge.net/projects/mysqlnavigator/), or Webmin with the module for working with MySQL (packages webmin-core and webmin-mysql) although the latter is no longer included with some distributions. Similarly to PostgreSQL, Webmin also permits working with MySQL (we will need to install the webmin-mysql packages as well as webmin-core). During the installation, webmin will warn that the main user will be the root and will use the same password as the root of the operating system. To connect, we can type, for example, https://localhost:10000 on the URL bar of a navigator which will request acceptance (or denial) of the use of a certificate for the SSL communication and next it will show all the services that can be administered, among them the MySQL Data Base Server.

MySQL Administrator is a powerful application for administering and controlling databases based on MySQL. This application integrates DB management, control and maintenance in a simple fashion and in the same environment. Its main characteristics are: advanced administration of large DBs, fewer errors through "visual administration", greater productivity and a safe management environment. The following figure shows a view of MySQL Administrator (in http://dev.mysql.com/doc/administrator/en/index.html we can find all of the documentation for installing it and starting it up).

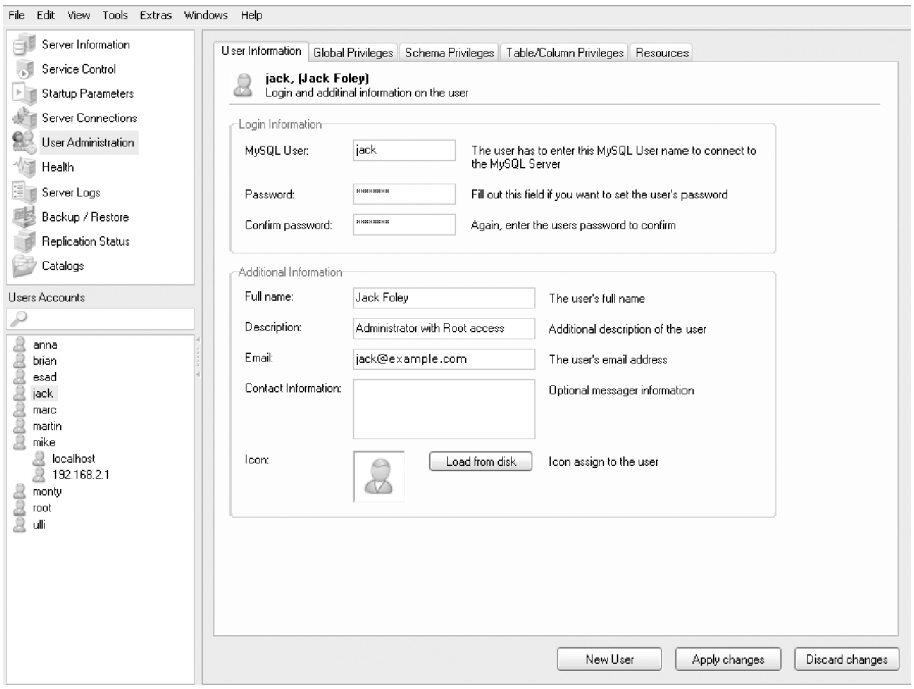

Figure 2. MySQL Administrator

3. Source Code management systems

The concurrent versions system (CVS) is a version control system that allows old version of files to be maintained (generally source code), saving a log of who made any changes, when and why. Unlike other systems, CVS does not work with a file/directory per occasion, but rather acts on hierarchical groups of the directories it controls.

The purpose of CVS is to help to manage software versions and to control the concurrent editing of source files by multiple authors. CVS uses another package called RCS (*revision control system*) internally as a low level layer. Although RCS can be used independently, it is not recommended, because in addition to its own functionality CVS offers all the capabilities of RCS but with notable improvements in terms of stability, functioning and maintenance. Among which we would highlight: decentralised operation (every user can have their own code tree), concurrent editing, adaptable behaviour through shell scripts etc. [Ced, CVS, Vasa, Kie]

As already explained in the introduction, Subversion (http://subversion.tigris.org/) is a version control system software specifically designed to replace the popular CVS, and to extend its capabilities. It is free software under an Apache/BSD type license and is also known as svn for the name on the command line. An important feature of Subversion is that unlike CVS the versioned files do not each have an independent revision number and instead the entire repository has a single version number which identifies a common status of all the repository's files at the time that it was "versioned". Among the main features we would mention:

- File and directory history can be followed through backups and renamings.
- Atomic and secure modifications (including changes to several files).
- Efficient and simple creation of branches and labels.
- Only the differences are sent in both directions (in CVS, complete files are always sent to the server).
- It can be served by Apache, over WebDAV/DeltaV.
- It handles binary files efficiently (unlike CVS, which treats them internally as if they were text).

There is an interesting free book that explains everything related to Subversion http://svnbook.red-bean.com/index.es.html and the translation is fairly advanced (http://svnbook.red-bean.com/nightly/es/index.html).

3.1. Revision control system (RCS)

Given that CVS is based on RCS and is still used on some systems, we will offer a few brief explanations of this software. RCS consists of a set of programs for its different RCS activities: `rcs` (program that controls file attributes under RCS), `ci` and `co` (which verify the entry and exit of files under RCS control), `ident` (searches RCS for files using key words/attributes), `rcsclean` (cleans files that are not used or that have not changed), `rcsdiff` (runs the diff command to compare versions), `rcsmerge` (joins two branches [files] into a single file), and `rlog` (prints log messages).

The format of the files stored by RCS can be text or another format, like binary for example. An RCS file consists of an initial revision file called 1.1 and a series of files of changes, one for each revision. Every time a copy of the repository is made to the work directory with the `co` (which obtains a revision of every RCS file and puts it in the work file) or `ci` (which stores new revisions in the RCS) commands is used, the version number is increased (for example, 1.2, 1.3,...). The files are (generally) in the /RCS directory and the operating system needs to have the `diff` and `diff3` commands installed in order for it to function properly. In Debian, it does not have to be compiled since it is included in the distribution.

With the rcs command we will create and modify file attributes (consult `rcs man`). The easiest way to create a repository is to create a directory with `mkdir rcs` in the directory of originals and include the originals in the repository using: `ci name_files_sources`.

We can use the `*` and should always have a backup copy to avoid problems. This will create the versions of the files with the name `./RCS/file_name` and request a descriptive text for the file. Then, using `co RCS/file_name`, we will obtain a work copy from the repository. This file can be blocked or unblocked to prevent modifications, respectively, using:

```
rcs -L workfile_name
rcs -U workfile_name
```

With `rlog file_name` we will be able to see the information on the different versions. [Kie]

3.2. Concurrent versions system (CVS)

First we need to install the concurrent versions system (CVS) from the distribution bearing in mind that we must have RCS installed and that we should also install OpenSSH if we wish to use it in conjunction with CVS for remote access. The environment variables EDITOR CVSROOT must also be initiated for example in /etc/profile (or in .bash profile):

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

Obviously, users can modify these definitions using /.bash profile. We need to create the directory for the repository and to configure the permissions; as root, we have to type, for example:

```
export CVSROOT = /usr/local/cvsroot
groupadd cvs
useradd -g cvs -d $CVSROOT cvs
mkdir $CVSROOT
chgrp -R cvs $CVSROOT
chmod o-rwx $CVSROOT
chmod ug+rwx $CVSROOT
```

To start up the repository and save the code file in it:

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

cvs init will take into account never overwriting an already created repository to avoid the loss of other repositories. Next, we will need to add the users that will work with CVS to the cvs group; for example, to add a user called *nteum*:

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

Now user *nteum* will have to save his or her files in the repository directory (/usr/local/cvsroot in our case) by typing:

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

The name of the repository can be a single identifier or also user/project/xxxx if the user wishes to have all their repositories organised. This will create a tree of directories in CVSROOT with that structure.

This adds a directory (/usr/local/cvsroot/RepositoryName) in the repository with the files that will be in the repository as of that moment. A test to know whether everything has been stored correctly is to save a copy in the repository and then create a copy from there and check the difference. For example, with the originals in user_directory /dir_org if we want to create a repository first_cvs/proj, we will have to execute the following commands:

`cd dir_org` Change to the original source code directory.

```
cvs import -m \
"Original sources"
\primer_cvs/proj userX vers0
```

Creates the repository in first_cvs/proj with userX and vers0.

`cd..` Change to the superior directory dir_org.

```
cvs checkout primer_cvs/proj
```

Generating a copy of the repository. The variable CVSROOT mustbe initiated, otherwise the full path will have to be shown.

```
diff -r dir_org primer_cvs/proj
```

Shows the differences between one and the other; there should not be any except for the directory first_cvs/proj/CVS created by CVS.

```
rm -r dir_org
```

Deletes the originals (always make a backup copy for safety and to have a reference of where the work with CVS started).

The following figure shows the organisation and how the files are distributed between versions and branches.

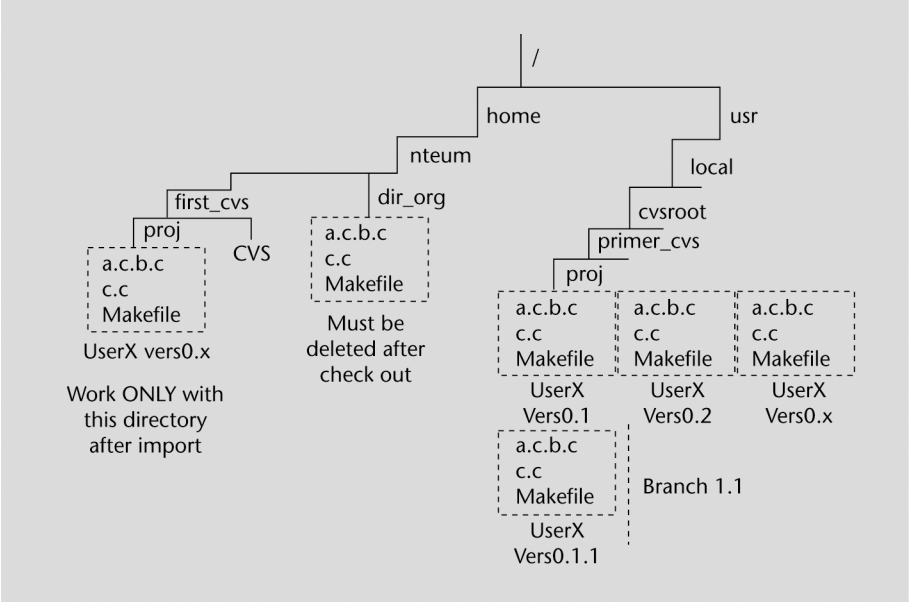

Figure 3

Deleting the originals is not always a good idea; only in this case, after verifying that they are in the repository, so that they will not be worked on by mistake and the changes will not be reflected on the CVS. On machines where users will want to access a remote CVS server (by ssh), we must type:

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

```
export CVS_RSH = "ssh"
```

Where user is the user login and cvs.server.com the name of the server with CVS. CVS offers a series of commands (named as cvs command options...) to work with the revision system, including: checkout, update, add, remove, commit and diff.

The initial cvs checkout command creates its own private copy of the source code so as to later work with it without interfering with the work of other users (at minimum it creates a subdirectory where the files will be located).

- `cvs update` must be executed from the private tree when copies of source files have to be updated with the changes made by other programmers to the repository's files.

- `cvs add file...` this command is necessary when we need to add new files to the work directory on a module that has already previously run a checkout. These files will be sent to the CVS repository when we execute the cvs commit command.

- `cvs import` can be used for introducing new files to the repository.

- `cvs remove file...` this command will be used to delete files from the repository (once these have been deleted from the private file). This command has to be accompanied by a cvs commit command for the changes to become effective, since this is the command that converts all of the users requests over the repository.

- `cvs diff file...` it can be used without affecting any of the files involved if we need to verify differences between repository and work directory or between two versions.

- `cvs tag -R "version"` can be used for introducing a version number in project files and then typing cvs commit and a `cvs checkout -r 'version'` project in order to register a new version.

An interesting characteristic of cvs is that it is able to isolate the changes to files isolated on a separate line of work called a branch. When we change a file on a branch, these changes do not appear on the main files or on other branches. Later, these changes can be incorporated to other branches or to the main file (merging). To create a new branch, use `cvs tag -b rel-1-0-patches` within the work directory, which will assign the name rel-1-0-patches to the branch. To join the branches to the work directory involves using the `cvs update -j` command. Consult references for merging or accessing different branches.

3.2.1. Example of a session

Following the example of the documentation provided in the references, we will show a work session (in a general form) with cvs. Since cvs saves all the files in a centralised repository, we will assume that it has already been initiated.

Let's suppose that we are working with a set of files in C and a *makefile*, for example. The compiler we use is gcc and the repository is initialised to gccrep.

In the first place, we will need to obtain a copy of the repository files as our own private copy with:

```
cvs checkout gccrep
```

This will create a new directory called gccrep with the source files. If we execute `cd gccrep` and `ls`, we will see, for example, cvs makefile a.c b.c c.c, where there is a cvs directory that is created to control the private copy that normally we do not need to touch.

We could use an editor to modify a.c and introduce substantial changes in the file (see the documentation on multiple concurrent users if we need to work with more than one user on the same file), compile, change again etc.

When we decide that we have a new version with all the changes made in a.c (or in the necessary files), it is time to make a new version by saving a.c (or all those that have been touched) in the repository and making this version available to the rest of the users: `cvs commit a.c.`

Using the editor defined in the variable CVSEDITOR (or EDITOR if it is not initialised) we will be able to enter a comment that discusses what changes have been made to help other users or to remind what characterised this version so that a log can be made.

If we decide to eliminate the files (because the project was completed or because it will not be worked on any more), one way of doing this is at the level of the operating system (rm -r gccrep), but it is better to use the cvs itself outside of the work directory (level immediately above): `cvs release -d gccrep`. The command will detect whether there is any file that has not been sent to the repository, and if there is and it is erased, it means that all the changes will be lost, which is why it will ask us if we wish to continue or not.

To look at the differences for example, b.c has been changed and we do not remember what changes were made, within the work directory, we can use: `cvs diff b.c`. This will use the operating system's diff command to compare

version b.c with the version in the repository (we must always remember to type `cvs commit b.c` if we want these differences to be transferred to the repository as a new version).

3.2.2. Multiple users

When more than one person works on a software project with different revisions, it is extremely complicated, because more than one user will sometimes want to edit the same file simultaneously. A potential solution is to block the file or to use verification points (reserved checkouts), which will only allow one user to edit the same file simultaneously. To do so, we must execute the command `cvs admin -l command` (see man for the options).

cvs uses a default model of unreserved checkouts, which allows users to edit a file in their work directory simultaneously. The first one to transfer their changes to the repository will be able to do so without any problems, but the rest will receive an error message when they wish to perform the same task, meaning that they must use cvs commands in order to transfer firstly the changes to the work directory from the repository and then update the repository with their own changes.

Consult the references to see an example of its application and other ways of working concurrently with communication between users. [Vasa].

3.3. Graphic interfaces

We have a set of graphic interfaces available such as tkcvs (http://www.twobarleycorns.net/tkcvs.html) [gcus] developed in Tcl/Tk and that supports subversion, or an also very popular one, cervisia [Cerc].

In the cvs wiki (http://ximbiot.com/cvs/wiki/index.php?title=CVS_Clients) we can also find a set of clients, plugins for cvs. Next, we will look at two of the mentioned graphic interfaces (tkcvs and gcvs):

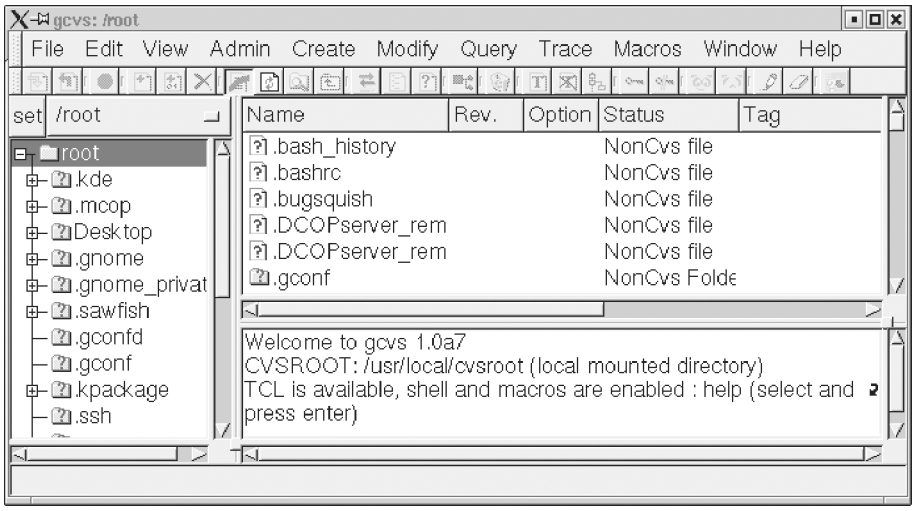

Figure 4. TkCVS (TkSVN)

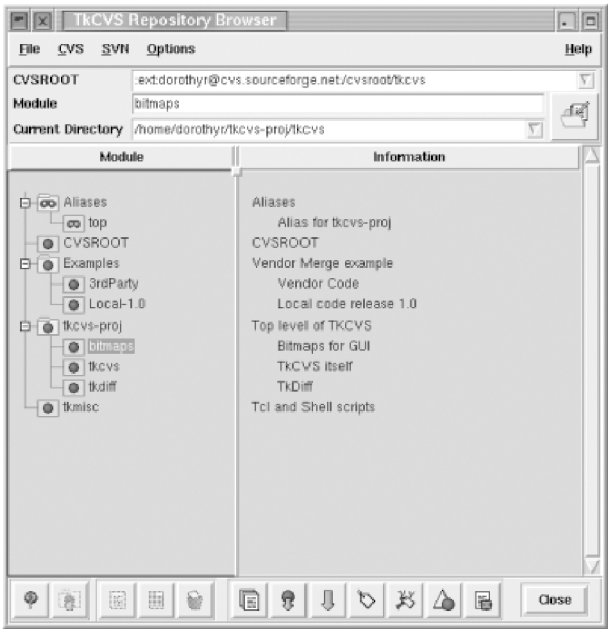

Figure 5. gCVS

4. Subversion

As an initial idea subversion serves to manage a set of files (repository) and its different versions. It is relevant to note that we do not care how the files are saved, but rather how we can access them, for which it is common to use a database. The idea of a repository is like a directory from which we want to recover a file of one week or 10 months ago based on the database status, to recover the latest versions and add new ones. Unlike cvs, subversion makes global revisions of the repository, which means that a change in a file does not generate a leap in version of that file only, but also of the entire repository, which adds one to the revision. In addition to the book we have mentioned (http://svnbook.red-bean.com/nightly/es/index.html), consult the documentation at http://subversion.tigris.org/servlets/ProjectDocumentList.

In Debian, we will have to type `apt-get install subversion`, if we wish to publish the repositories in Apache2 `apt-get install Apache2-common` and the specific module `apt-get install libApache2-`subversion.

- First step: create our repository, user (we assume the user is svuser), group (svgroup) as *root*...
  ```
  mkdir -p /usr/local/svn
  addgroup svgroup
  chown -R root.svgroup /usr/local/svn
  chmod 2775 /usr/local/svn
  ```

- `addgroup svuser svggroup` Adds the svuser user to the svgroup group.

- We connect as svuser and verify that we are in the svgroup group (with the group command).

- `svnadmin create /usr/local/svn/tests`
  This command will create a series of files and directories for version management and control. If we are not permitted in /usr/local/svn, we can do so in the local directory: `mkdir -p $HOME/svndir` and next `svnadmin create $HOME/svndir/tests.`

- Next we create a temporary directory `mkdir -p $HOME/svntmp/tests` we move to the directory cd $HOME/svntmp/tests and create a file like: `echo First File Svn 'date' > file1.txt.`

- We transfer it to the repository: inside the directory we type `svn import file:///home/svuser/svndir/tests-m "View. Initial"`. If we have created it in `/usr/local/svn/tests` we should type the full path after `file://`. The import command copies the directory tree and

the -m option allows the version message to be shown. If we do not add the -m option, an editor will open to do so (we need to enter a message in order to avoid problems). The subdirectory $HOME/svntmp/tests is a copy of the work in the repository and deleting it is recommended so as not to be tempted to commit the error of working with it and not with the repository (`rm -rf $HOME/svntmp/tests`).

- Once in the repository, we can obtain the local copy where we can work and then upload the copies to the repositories, by typing:
  ```
  mkdir $HOME/svm-work
  cd $HOME/svn-work
  svn checkout file:///home/svuser/svndir/tests
  ```
  Where we will see that we have the tests directory. We can copy with another name adding the name we want at the end. To add a new file to it:
  ```
  cd /home/kikov/svn-work/tests
  echo Second File Svn 'date' > file2.txt
  svn add file2.txt
  svn commit -m"New file"
  ```
  It is important to note that once in the local copy (svn-work) we must not specify the path. svn add marks to add the file to the repository and that really it is added when we run cvn commit. It will give us some messages indicating that it is the second version.

If we add another line file1.txt with `echo 'date'>>file1.txt`, then we will be able to upload the changes with: `svn commit -m"New line"`.

It is possible to compare the local file with the repository file, for example we add a third line to file1.txt with `echo 'date'>>file1.txt`, but we do not upload it and if we want to see the differences we can run: svn diff.

This command will highlight what the differences are between the local file and those of the repository. If we load it with `svn commit -m"New line2"` (which will generate another version) then the svn diff will not give us any differences.

We can also use the command svn update within the directory to update the local copy. If there are two or more users working at the same time and each one has made a local copy of the repository and modifies it (by doing commit), when the second user goes to commit their copy with their modifications, they will receive a conflict error, since the copy in the repository has a more recent modification date than this user's original copy (in other words, there have been changes in between), meaning that if the second user runs commit, we could lose the modifications of the first one. To do this, we must run svn update which will tell us the file that creates a conflict with a C and will show us the files where the parts in conflict have been placed. The user must decide what version to keep and whether they can run commit.

An interesting command is the `svn log file1.txt`, which will show all the changes that have been made to the file and its corresponding versions.

An interesting feature is that subversion can run in conjunction with Apache2 (and also over SSL) to be accessed from another machine (consult the clients in http://svnbook.red-bean.com/) or simply look at the repository. In Debian Administration they explain how to configure Apache2 and SSL for Sarge, or as we already indicated in the part on servers. For this, we need to activate the WebDAV modules (see http://www.debian-administration.org/articles/285 or in their absence http://www.debian-administration.org/articles/208.

As root user we type:
`mkdir /subversión chmod www-data:www-data`

So that Apache can access the directory
`svnadmin create /subversion`

we create the repository
`ls -s /subversion`

```
-rw-r--r-- 1 www-data www-data 376 May 11 20:27 README.txt
drwxr-xr-x 2 www-data www-data 4096 May 11 20:27 conf
drwxr-xr-x 2 www-data www-data 4096 May 11 20:27 dav
drwxr-xr-x 2 www-data www-data 4096 May 11 20:28 db
-rw-r--r-- 1 www-data www-data 2 May 11 20:27 format
drwxr-xr-x 2 www-data www-data 4096 May 11 20:27 hooks
drwxr-xr-x 2 www-data www-data 4096 May 11 20:27 locks
```

For authentication we use htpasswd (for example with `htpasswd2 -c -m /subversion/.dav_svn.passwd user` created as www-data. We only have to type the -c the first time that we execute the command to create the file. This tells us that in order to access this directory we need a password (which is the one we have entered for user).

Then we will need to change the httpd.conf so that it is something like:

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

We reinitiate Apache and now we are ready to import some files, such as:

```
svn import file1.txt http://url-server.org/svn \
-m "Import Initial"
```

We will be asked for authentication (user/password) and told that the file file1.txt has been added to the repository.

Activities

1) Define in PostgreSQL a DB that has at least 3 tables with 5 columns (of which 3 must be numerical) in each table.

Generate an ordered list for each table/column. Generate a list ordered by the highest value of the *X* column of all tables. Change the numerical value in the *Y* column with the numerical value of column *Z* + the value of column *W*/2.

2) The same exercise as above, but with MySQL.

3) Configure the cvs to make three revisions of a directory where there are 4 .c files and a makefile. Make a branch of the file and then merge it with the main one.

4) Simulate the concurrent use of a file with two Linux terminals and indicate the sequence of steps to be done so that the two alternating modifications of each user are reflected in the cvs repository.

5) Same exercise as above, but one of the users must connect to the repository from another machine.

6) Idem 3, 4 and 5 in Subversion.

Bibliography

Other sources of reference and information

[Debc, Ibi, Mou01]

PgAccess: http://www.pgaccess.org/

WebMin: http://www.webmin.com/

Mysql Administrator http://www.mysql.com/products/tools/administrator/

Graphic interfaces for CVS: http://www.twobarleycorns.net/tkcvs.html

Or in the CVS wiki: http://ximbiot.com/cvs/wiki/index.php?title=CVS_Clients

Subversion: http://subversion.tigris.org

Free Book about Subversion: http://svnbook.red-bean.com/index.es.html

Apache2 and SSL: http://www.debian-administration.org/articles/349

Apache2 and WebDav: http://www.debian-administration.org/articles/285

There is a large amount of documentation about Apache and SSL + Subversion in Debian as well as http://www.debian-administration.org, in Google, enter "Apache2 SSL and Subversion in Debian" to obtain some interesting documents.