73 - PHP-phpMyAdmin -MySQL

LAMP - Linux-Apache-MySQL-PHP

1) Install the following packages:

```
apache2                 mysql               phpMyAdmin
apache2-prefork         mysql-client        php5-mysql
apache2-mod_php5        mysql-shared        php5
```

2) Create a Virtualhost pointing to `/srv/www/htdocs/phpMyAdmin` directory including the Directives:

```
NameVirtualHost  192.168.71.10 (only if doesn't already exist)
<VirtualHost 192.168.71.10>
     ServerName phpmyadmin.linux.site
     DocumentRoot /srv/www/htdocs/phpMyAdmin
     <Directory /srv/www/htdocs/phpMyAdmin>
          Allow from all
          DirectoryIndex  index.php
     </Directory>
</VirtualHost>
```

3) Insert the local IP and server name in `/etc/hosts`

eg. `129.168.71.10  phpmyadmin.linux.site`

4) Start the msql daemon with the command:

```
rcmysql start
```

5) Change the root password in mysql using the commands:

```
mysqladmin –u root password newpasswd
mysqladmin –u root–h hostname.linux.site password newpasswd
```

Press <Enter> when asked for the password.

6) Change the root password and Absolute URI in
`/srv/www/htdocs/phpMyAdmin/config.inc.php`
eg.

```
$cfg['PmaAbsoluteUri'] = 'http://phpmyadmin.linux.site';
$cfg['Servers'][$i]['user']     = 'root';      // MySQL user
$cfg['Servers'][$i]['password'] = 'newpasswd'; // MySQL password
```

7) Start or Restart Apache Server.

```
rcapache2 restart
```

8) Start a browser and write the address:

```
http://phpmyadmin.linux.site
```

Extra Tips:
Command to create a new database in mysql client:

```
> create database DatabaseName;
```

WAMP-Windows-Apache-MySQL-PHP

Installing Apache2, PHP4 and MySQL in Windows2000/XP

There is a web site where you can download an installer which installs all the components of Apache2, PHP4, Mysql, PhpMyAdmin, Webalizer and extras automatically at:
`http://www.apachefriends.org/de/xampp-windows.html`
The lastest version at the moment of writing was: V. 1.4.3
Just run the file and all will be done automatically.

Manual installation for the brave!

Note: This installation cuts corners and does things that might not be necessary.
I've used this method and got a working setup.
For other alternatives, please refer to the document:
`http://www.php.net/manual/en/install.apache2.php`

INSTALLING APACHE2

1) Download the latest stable Apache for Windows from the internet:
`http://www.apache.org`
It's a `.msi` file and can be installed immediately

2) In my case, I kept the recommended install path of Apache:
`C:/programm(e)/Apache Group/Apache2/`

INSTALLING MYSQL

1) Download the MySQL windows installer from the Internet:
`http://dev.mysql.com/downloads/mysql/4.0.html`
It's a ZIP file
2) Unpack the ZIP file into a temporary directory and run the program `SETUP.EXE`
I installed it in `C:\mysql` directory

INSTALLING PHP4(as apache module)

1) Download the ZIP file from:
`http://www.php.net/downloads.php`
2) Unzip the files into `C:/php` directory
3) copy all the `.dll` files from `C:/php/dlls` and `C:/php/sapi` to `C:/php` directory

INSTALLING phpMyAdmin

1) Download the latest version of phpMyAdmin ZIP file from:
`http://www.phpmyadmin.net/home_page/`
2) Unzip the file into the directory:
`C:\Programme\Apache Group\Apache2\htdocs\phpmyadmin\`
3) Edit the `config.inc.php` in the `phpmyadmin` directory and:
- add the proper full URI of the phpMyAdmin:
`$cfg['PmaAbsoluteUri'] = 'http://localhost/phpmyadmin/';`

OPTIONAL:(If you leave the root password blank in MySQL then leave it blank here also)
- add the MySQL `root` (administrator) password in clear text in:

```
$cfg['Servers'][$i]['user']          = 'root';  // MySQL user
$cfg['Servers'][$i]['password']      = '';      // MySQL password
                          (only needed with'config' auth_type)
```

CONFIGURATION

1) Edit the `httpd.conf` file and:
- Change to the follwoing settings to:
```
DocumentRoot "C:/Programme/Apache Group/Apache2/htdocs"
DirectoryIndex index.html index.html.var index.php
```
- Add the following settings:
```
LoadModule php4_module c:/php/php4apache2.dll
LoadFile C:/php/php4ts.dll
AddType application/x-httpd-php .php .php3 .php4 .phtml
```

2) Edit the `C:\php\php.ini` file
- You will need to change the `extension_dir` setting to point to your `php-install-dir`, or where you have placed your `'php_*.dll' files`. eg: `c:\php`
  It is located around the line 442
- Set the `'doc_root'` to point to your webservers `document_root`.
  in my case: `C:/Programme/Apache Group/Apache2/htdocs`

3) Create the file `C:\MY.CNF` and insert the following lines in it:
```
[mysqld]
basedir=C:/mysql/
datadir=C:/mysql/data/
```

Note: See the MY.CNF example from XAMPP below

OPERATING WAMP

1) Open a DOS window `(cmd)` and give the command:
```
C:\mysql\bin\mysqld.exe
```
When all is ok then an icon can be created to start it either from the autostart area or by hand.

2) start Apache through the Windows start menu

3) Start a browser and give the address:
```
http://localhost/phpmyadmin/
```

You should have a phpMyAdmin web site with full control over the MySQL databases. This would confirm that your Apache, PHP and MySql are all working...so far go good!

MY.CNF (Example from XAMPP)

```
# The MySQL client
[client]
#password       = your_password
port            = 3306
socket          =c:/WAMP/xampp/mysql/mysql.sock

# Here follows entries for some specific programs

# The MySQL server
[mysqld]
port            = 3306
socket          = c:/WAMP/xampp/mysql/mysql.sock
skip-locking
set-variable    = key_buffer=16M
set-variable    = max_allowed_packet=1M
set-variable    = table_cache=64
set-variable    = sort_buffer=512K
set-variable    = net_buffer_length=8K
set-variable    = myisam_sort_buffer_size=8M
log-bin
server-id       = 1

basedir=c:/WAMP/xampp/mysql
tmpdir=c:/WAMP/xampp/tmp
datadir=c:/WAMP/xampp/mysql/data

#bind-address=192.168.1.1
#log-update     = /path-to-dedicated-directory/hostname

# Uncomment the following if you are using BDB tables
#set-variable   = bdb_cache_size=4M
#set-variable   = bdb_max_lock=10000

skip-innodb

# Uncomment the following if you are using InnoDB tables
#innodb_data_home_dir = c:/WAMP/xampp/mysql/
#innodb_data_file_path = ibdata1:10M:autoextend
#innodb_log_group_home_dir = c:/WAMP/xampp/mysql/
#innodb_log_arch_dir = c:/WAMP/xampp/mysql/
# You can set .._buffer_pool_size up to 50 - 80 %
# of RAM but beware of setting memory usage too high
#set-variable = innodb_buffer_pool_size=16M
#set-variable = innodb_additional_mem_pool_size=2M
# Set .._log_file_size to 25 % of buffer pool size
#set-variable = innodb_log_file_size=5M
#set-variable = innodb_log_buffer_size=8M
#innodb_flush_log_at_trx_commit=1
#set-variable = innodb_lock_wait_timeout=50
```

```
[mysqldump]
quick
set-variable    = max_allowed_packet=16M

[mysql]
no-auto-rehash
```
Remove the next comment character if you are not familiar with SQL
```
#safe-updates
[isamchk]
set-variable    = key_buffer=20M
set-variable    = sort_buffer=20M
set-variable    = read_buffer=2M
set-variable    = write_buffer=2M

[myisamchk]
set-variable    = key_buffer=20M
set-variable    = sort_buffer=20M
set-variable    = read_buffer=2M
set-variable    = write_buffer=2M

[mysqlhotcopy]
interactive-timeout

[WinMySQLadmin]
Server=c:/WAMP/xampp/mysql/bin/mysqld-nt.exe
```

mysql & mysql-client

- Install `mysql` and `mysql-client` and `knoda` from SuSE CD/DVD via Yast.
- In SuSE till 7.3 :Set `START_MYSQL=yes` and `HTTPD_SEC_MOD_PHP=yes` ....in Yast
- In SuSE 8.0 and 8.1: Start run level editor(yast2) and add mysql in run levels 3 & 5

Running mysql

- Start mysql server: `rcmysql start`
- Change the `root` password of `mysql` server via the command:

```
mysqladmin -u root -p password 'new-password'
mysqladmin -u root -h p1.linux.local -p password 'new-password'
```

(Only press on ENTER when password asked here)

Info:

- mysql daemon user: `mysql`
- TCP Port `3306`

Files: `/usr/bin/mysql_install_db`

Script started once when the server is started for the first time.
It installs the databases and instruct the user on how to change the mysql root password.

Monitoring MySQL's well working via mysql client:

`mysql -p`	- Starts the monitor program in terminal and enter the mysql root passsword
mysql> **`show databases;`**	- Displays all the MySQL Databases
mysql> **`use mysql;`**	- Uses the `mysql` database
mysql> **`show tables;`**	- Displays the tables of `mysql` database
mysql> **`select * from user \G;`**	- Displays `user` table form `mysql` database
mysql> **`quit`**	

phpMyAdmin

- Install `phpMyAdmin` from SuSE CD
- Copy the phpMyAdmin directory content to the Virtual Host's `DocumentRoot`'s
- Set the DirectoryIndex of VirtualHost to `index.php`
- Edit the file `config.inc.php` in the (in `phpMyAdmin` directory)
  Change the root password the the one given above for mysql server ~Line 49
  `$cfgServers[1]['password'] = 'mysql-root-password';`
- make sure this file(`config.inc.php`) is not readable from
- Make sure that PHP4 Module is started in Apache (`/etc/sysconfig/apache`)
- Make sure Apache recognizes the `.php3` files as running under php4 module. Edit Apache configuration file(`/etc/httpd/httpd.conf`)

```
DirectoryIndex .....Index.php3....
AddApplication ....... .php3
```

- Restart Apache (`rcapache restart`)
- In browser `http://VirtualHostName/`

Language:

-To run older phpMyAdmin in German, just edit the line 57 in `config.inc.php3` in phpMyAdmin directory and change it from :

`require("english.inc.php3");` to `require("german.inc.php3");`

Note: the new phpMyAdmin from SuSE 8.0 is version 2.2.3 and allows the language to be changed via the interface.

Security:

1) When phpMyAdmin is logged onto MySQL using a `mysql` username that has all the access rights, it can disturb a lot in other's databases.

To avoid this:

- Make a copy of phpMyAdmin in each user's `/public_html` directory
- Edit the file `config.inc.php3` in `phpMyAdmin` directory to include only the user's access name and password as default.
- Add the user its password and access rights to mysql database
- Make sure that the file `config.inc.php3` is only writable by `root`
- Let the user be authenticated for access to his `phpMyAdmin` directory via Apache authentication.
- Access the phpMyAdmin management via
  `http://servername/~username/phpMyAdmin/`
- Locally the MySQL databases are located in `/var/lib/mysql`

2) When the phpMyAdmin is installed the file `config.inc.php` is readable by all. This is a security risk since the file has the mysql root password in clear text format. To avoid this:

- Make the file owned by apache user (wwwrun) and access rights of 400

New user of the Mysql database:

- Click on `+mysql` on left panel
- Click on `Browse` in `user` line
- Click on `insert new row`
- Enter:
  - Host Address or name
  - Name of user
  - At `Password` field:
    - Click on button in `Function` column
    - Select `PASSWORD`
    - Enter the password in clear text
    - Select `Y` or `N` for each SQL command allowed or not by the user.
    - Click on `Save` Button
  - To change the user's access rights click on `edit` at the end of user's line in the users table.

-The user accounts that phpMyAdmin uses to login to MySQL are set in `config.inc.php` in phpMyAdmin directory.

What if:

1) Situation: I can't log-in as `root` user in MySQL server,
or I have forgotten its root password.

Solution:

If running MySQL on SuSE Distribution:
as Root:

- Stop the mysql server
```
rcmysql stop
```

- Start the mysql server with options to avoid using the users rights granting table.

```
/usr/bin/safe_mysqld --user=mysql --datadir=/var/lib/mysql \
     --skip-grant-tables
```

- Start mysql client and empty the old password
```
mysql -u root
mysql> USE mysql;
mysql> UPDATE user SET password=''
     > WHERE user='root' AND host='localhost';
mysql> quit
```

- Restart the Mysql server with normal operation
```
rcmysql restart
```

- Finally change the empty root password to a new one
```
mysqladmin -u root -h localhost password 'newpassword'
```

2) Exporting/Importing a database into SQL Format (Coma separated text data)

Exporting to SQL format file? issue the following command:
```
mysqldump -u root -h localhost --password=Password --opt\
                                   DatabaseName > SQLFilename
```
Import from the SQL File into Database:
```
echo "create database DatabaseName"| mysql -u root -h localhost
mysql -u root -h localhost DatabaseName < SQLFilename
```

3) Creating a MySQL database restricted to a specific new user.

Using the MySQL Client program:

1 - Create the new user and password:

Using mysql client:
```
mysql -u root -p
      (Enter the mysql root password)
mysql> GRANT usage ON *.* TO username@localhost
    > IDENTIFIED BY 'password';
```

2 - Assign access/modify privileges of database to new user:

Using MySQL client:

Issue the 2 commands:

```
INSERT INTO `db` (`Host`, `Db`, `User`, `Select_priv`,
`Insert_priv`, `Update_priv`, `Delete_priv`, `Create_priv`,
`Drop_priv`, `Grant_priv`, `References_priv`, `Index_priv`,
`Alter_priv`, `Create_tmp_table_priv`, `Lock_tables_priv`) VALUES
('localhost', 'test1', 'user5', 'Y', 'Y', 'Y', 'Y', 'Y', 'Y', 'Y',
'Y', 'Y', 'Y', 'Y', 'Y');
```

`FLUSH PRIVILEGES ;` (Needed to make the new access rights active)

Or Using phpMyAdmin:

Version: `2.7.0-rc1 on Debian Sarge 1`

1 – Create a new user with NO rights at all.

Database: `mysql`

Table: `user`

---> Insert New Row

2 – Create a new Database:

- `phpMyAdmin 2.7.0-rc1` with MySQL Version `4.0.24_Debian-10sarge1`
- Create a new empty database. eg. `testdb`

---> Insert a new row in the:

- Database: `mysql`
  - Table: `db ---- > Browse`

`----> Insert new row`

- Enter the values in the 3 first fields:
- Host: [`localhost`]
- Db: [*`testdb`*]
- User: [*`username`*]
- and select 'Y' for all of the first privileges group.

Note: No changes to privileges after the `ignore` selection should be done.

- Click on SQL tab and issue the SQL command:

`FLUSH PRIVILEGES ;` (Needed to make the new access rights active).

3 - The user can now fully manipulate this database when correctly logged in.

4 - Creating an account for the debian maintenance user:

Used to start stop of mysql and also to prevent the following annoying message:

```
Starting MySQL database server: mysqld.
/usr/bin/mysqladmin: connect to server at 'localhost' failed
error: 'Access denied for user 'debian-sys-maint'@'localhost' (using password: YES)'
```

Solution:

`> mysql -u root -p` (Give the root password)

```
> GRANT ALL PRIVILEGES ON *.* TO 'debian-sys-maint'@'localhost' IDENTIFIED BY
  '<password>' WITH GRANT OPTION;
```

(Change the password with the one taken from `/etc/mysql/debian.cnf`)

```
> FLUSH PRIVILEGES;
```

If done through phpMyAdmin then use the password with the one taken from `/etc/mysql/debian.cnf` and set it to OLD-PASSWORD field type and also do a `FLUSH PRIVILEGES;`

Creating a second instance of MySQL server on port 3307.

- Create a copy of the admin config file `/etc/mysql/debian.cnf` to `/etc/mysql/debian2.cnf` with the following content:

```
# Automatically generated for Debian scripts. DO NOT TOUCH!
[client]
host     = localhost
user     = debian-sys-maint
password = 1cddmCxKskBGzzA5
socket   = /var/run/mysqld/mysqld2.sock

[mysql_upgrade]
user     = debian-sys-maint
password = 1cddmCxKskBGzzA5
#password = piP7DUoNVeG7R2Pe
socket   = /var/run/mysqld/mysqld2.sock
basedir  = /usr
```

- Create a copy mysql main config file `/etc/mysql/my.cnf` to `/etc/mysql/my.cnf` and change the port, socket, daemon ect. as follows:

```
[client]
port            = 3307
socket          = /var/run/mysqld/mysqld2.sock

[mysqld]
#
# * Basic Settings
#
user            = mysql
pid-file        = /var/run/mysqld/mysqld2.pid
socket          = /var/run/mysqld/mysqld2.sock
port            = 3307
basedir         = /usr
datadir         = /var/lib/mysql2
.....................
.....................      the rest can stay the same.
```

- Create a copy of the file `debian-start` to `debian-start2` with the following content:

```
#!/bin/bash
#
# This script is executed by "/etc/init.d/mysql" on every (re)start.
#
# Changes to this file will be preserved when updating the Debian package.
#

source /usr/share/mysql/debian-start.inc.sh

MYSQL="/usr/bin/mysql --defaults-file=/etc/mysql/debian2.cnf"
MYADMIN="/usr/bin/mysqladmin --defaults-file=/etc/mysql/debian2.cnf"
MYUPGRADE="/usr/bin/mysql_upgrade --defaults-extra-file=/etc/mysql/debian2.cnf"
MYCHECK="/usr/bin/mysqlcheck --defaults-file=/etc/mysql/debian2.cnf"
MYCHECK_SUBJECT="WARNING: mysqlcheck has found corrupt tables"
MYCHECK_PARAMS="--all-databases --fast --silent"
MYCHECK_RCPT="root"

# The following commands should be run when the server is up but in background
```

```
# where they do not block the server start and in one shell instance so that
# they run sequentially. They are supposed not to echo anything to stdout.
# If you want to disable the check for crashed tables comment
# "check_for_crashed_tables" out.
# (There may be no output to stdout inside the background process!)
echo "Checking for corrupt, not cleanly closed and upgrade needing tables."
(
  upgrade_system_tables_if_necessary;
  check_root_accounts;
  check_for_crashed_tables;
) >&2 &

exit 0
```

- Create a copy of the MySQL init script `/etc/init.d/mysql` to `/etc/init.d/mysql2` and modify the following at the beginning of the file:

```
CONF=/etc/mysql/my2.cnf
# Modified by michel to use another defauls file (/etc/mysql/debian2.cnf
--port=3307)
MYADMIN="/usr/bin/mysqladmin --defaults-file=/etc/mysql/debian2.cnf --port=3307"

# priority can be overriden and "-s" adds output to stderr
# Modified by michel: /etc/init.d/mysql2 -i
ERR_LOGGER="logger -p daemon.err -t /etc/init.d/mysql2 -i"
.........

# Usage: void mysqld_get_param option
# Modified by michel to use the config file of second instance of mysql
(/etc/mysql/my2.cnf)
mysqld_get_param() {
        /usr/sbin/mysqld --defaults-file=/etc/mysql/my2.cnf --print-defaults \
                | tr " " "\n" \
                | grep -- "--$1" \
                | tail -n 1 \
                | cut -d= -f2
}

## Do some sanity checks before even trying to start mysqld.
sanity_checks() {
  # check for config file
  # Modified by michel : /etc/mysql/my2.cnf
  if [ ! -r /etc/mysql/my2.cnf ]; then
```

- Create a symlink from `/usr/sbin/mysqld` called `/usr/sbin/mysqld2` using the command:
  `ln -s /usr/sbin/mysqld /usr/sbin/mysqld2`
- Make a copy of empty dadabases in `/var/lib/mysql2`. Originals are in:
  `hp1:/root/mysql_data_backups/mysql.empty`
- Give ownership of the databases to `mysql` user using the command:
  `chown -R mysql. /var/lib/mysql2`
- Enter the new init script in default runlevels with the comand:
  `update-rc.d mysql2 defaults`
- [Optional] Create a symlink for manual start/stop with the command:
  `ln -s /etc/init.d/mysql2 /usr/sbin/rcmysql2`
- Start the new mysql database with the command:
  `rcmysql start`

Live Backups of MySQL Using Replication

by Russell Dyer, author of MySQL in a Nutshell
06/16/2005

One of the difficulties with a large and active MySQL database is making clean backups without having to bring the server down. Otherwise, a backup may slow down the system and there may be inconsistency with data, since related tables may be changed while another is being backed up. Taking the server down will ensure consistency of data, but it means interruption of service to users. Sometimes this is necessary and unavoidable, but daily server outages for backing up data may be unacceptable. A simple alternative method to ensure reliable backups without having to shut down the server daily is to set up replication for MySQL.

Typically replication is a system configuration whereby the MySQL server, known in this context as a master server, houses the data and handles client requests, while another MySQL server (a slave server) contains a complete copy of the data and duplicates all SQL statements in which data is changed on the master server right after it happens. There are several uses for replication (e.g., load balancing), but the concern of this article has to do with using replication for data backups. You can set up a separate server to be a slave and then once a day turn replication off to make a clean backup. When you're done, replication can be restarted and the slave will automatically query the master for the changes to the data that it missed while it was offline. Replication is an excellent feature and it's part of MySQL. You just need to set it up.

The Replication Process

Before explaining how to set up replication, let me quickly explain the steps that MySQL goes through to maintain a replicated server. The process is different depending on the version of MySQL. For purposes of this article, my comments will be for version 4.0 or higher, since most systems now are using the later versions.

When replication is running, basically, as SQL statements are executed on the master server, MySQL records them in a binary log (`bin.log`) along with a log position identification number. The slave server in turn, through an IO thread, regularly and very often reads the master's binary log for any changes. If it finds a change, it copies the new statements to its relay log (`relay.log`). It then records the new position identification number in a file (*master.info*) on the slave server. The slave then goes back to checking the master binary log, using the same IO thread. When the slave server detects a change to its relay log, through an SQL thread the slave executes the new SQL statement recorded in the relay log. As a safeguard, the slave also queries the master server through the SQL thread to compare its data with the master's data. If the comparison shows inconsistency, the replication process is stopped and an error message is recorded in the slave's error log (*error.log*). If the results of the query match, the new log position identification number is recorded in a file on the slave (`relay-log.info`) and the slave waits for another change to the relay log file.

This process may seem involved and complicated at first glance, but it all occurs quickly, it isn't a significant drain on the master server, and it ensures reliable replication. Also, it's surprisingly easy to set up. It only requires a few lines of options to be added to the configuration file (i.e., `my.cnf`) on the master and slave servers. If you're dealing with a new server, you'll need to copy the databases on the master server to the slave to get it caught up. Then it's merely a matter of starting the slave for it to begin replicating.

The Replication User

There are only a few steps to setting up replication. The first step is to set up a user account to use only for replication. It's best not to use an existing account for security reasons. To do this, enter an SQL statement like the following on the master server, logged in as root or a user that has `GRANT OPTION` privileges:

```
GRANT REPLICATION SLAVE, REPLICATION CLIENT
    ON *.*
    TO 'replicant'@'slave_host'
    IDENTIFIED BY 'my_pwd';
```

In this SQL statement, the user account *replicant* is granted only what's needed for replication. The user name can be almost anything. The host name (or IP address) is given in quotes. You should enter this same statement on the slave server with the same user name and password, but with the master's host name or IP address. This way, if the master fails and will be down for a while, you could redirect users to the slave with DNS or by some other method. When the master is back up, you can then use replication to get it up to date by temporarily making it a slave to the former slave server.
Incidentally, if you upgraded MySQL to version 4.0 recently, but didn't upgrade your `mysql` database, the `GRANT` statement above won't work because these privileges didn't exist in the earlier versions. For information on fixing this problem, see MySQL's documentation on Upgrading the Grants Tables.

Configuring the Servers

Once the replication user is set up on both servers, we will need to add some lines to the MySQL configuration file on the master and on the slave server. Depending on the type of operating system, the file will probably be called `my.cnf` or `my.ini`. On Unix-type systems, the configuration file is usually located in the */etc* directory. On Windows systems, it's usually located in *c:* or in *c:\Windows*. Using a text editor, add the following lines to the configuration file, under the `[mysqld]` group heading:

```
server-id = 1
log-bin = /var/log/mysql/bin.log
```

The server identification number is an arbitrary number to identify the master server. Almost any whole number is fine. A different one should be assigned to the slave server to keep them straight. The second line above instructs MySQL to perform binary logging to the path and file given. The actual path and file name is mostly up to you. Just be sure that the directory exists and the user *mysql* is the owner, or at least has permission to write to the directory. Also, for the file name use the suffix of ".log" as shown here. It will be replaced automatically with an index number (e.g., ".000001") as new log files are created when the server is restarted or the logs are flushed.

For the slave server, we will need to add a few more lines to the configuration file. We'll have to provide information on connecting to the master server, as well as more log file options. We would add lines similar to the following to the slave's configuration file:

```
server-id = 2

master-host = mastersite.com
master-port = 3306
master-user = replicant
master-password = my_pwd
```

```
log-bin = /var/log/mysql/bin.log
log-bin-index = /var/log/mysql/log-bin.index
log-error = /var/log/mysql/error.log

relay-log = /var/log/mysql/relay.log
relay-log-info-file = /var/log/mysql/relay-log.info
relay-log-index = /var/log/mysql/relay-log.index
```

This may seem like a lot, but it's pretty straightforward once you pick it apart. The first line is the identification number for the slave server. If you set up more than one slave server, give them each a different number. If you're only using replication for backing up your data, though, you probably won't need more than one slave server. The next set of lines provides information on the master server: the host name as shown here, or the IP address of the master may be given. Next, the port to use is given. Port 3306 is the default port for MySQL, but another could be used for performance or security considerations. The next two lines provide the user name and password for logging into the master server.

The last two stanzas above set up logging. The second to last stanza starts binary logging as we did on the master server, but this time on the slave. This is the log that can be used to allow the master and the slave to reverse roles, as mentioned earlier. The binary log index file (*log-bin.index*) is for recording the name of the current binary log file to use. As the server is restarted or the logs are flushed, the current log file changes and its name is recorded here. The `log-error` option establishes an error log. If you don't already have this set up, you should, since it's where any problems with replication will be recorded. The last stanza establishes the relay log and related files mentioned earlier. The relay log makes a copy of each entry in the master server's binary log for performance's sake, the `relay-log-info-file` option names the file where the slave's position in the master's binary log will be noted, and the relay log index file is for keeping track of the name of the current relay log file to use for replicating.

Copying Databases and Starting Replication

If you're setting up a new master server that doesn't contain data, then there's nothing left to do but restart the slave server. However, if you're setting up replication with an existing server that already has data on it, you will need to make an initial backup of the databases and copy it to the slave server. There are many methods to do this; for our examples, we'll use the utility `mysqldump` to make a backup while the server is running. However, there's still the problem with attaining consistency of data on an active server. Considering the fact that once you set up replication you may never have to shut down your server for backups again, it might be worth while at least to lock the users out this one last time to get a clean, consistent backup. To run the master server so that only *root* has access, we can reset the variable `max_connections` like so:

```
SHOW VARIABLES LIKE 'max_connections';

+-----------------+-------+
| Variable_name   | Value |
+-----------------+-------+
| max_connections | 100   |
+-----------------+-------+

SET GLOBAL max_connections = 0;
```

The first SQL statement isn't necessary, but we may want to know the initial value of the `max_connections` variable so that we can change it back when the backup is finished. Although setting the variable to a value of `0` suggests that no connections are allowed, one connection is actually reserved for the *root* user. Of course, this will only prevent any new connections. To see if there are any connections still running, enter `SHOW PROCESSLIST;`. To terminate any active processes, you can use the `KILL` statement.

With exclusive access to the server, using `mysqldump` is usually very quick. We would enter the following from the command line on the master server:

```
mysqldump --user=root --password=my_pwd \
      --extended-insert --all-databases \
      --master-data  > /tmp/backup.sql
```

This will create a text file containing SQL statements to create all of the databases and tables with data. The `--extended-insert` option will create multiple-row `INSERT` statements and thereby allow the backup to run faster, for the least amount of down time or drain on services. The `--master-data` option above locks all of the tables during the dump to prevent data from being changed, but allows users to continue reading the tables. With exclusive access, this feature isn't necessary. However, this option also adds a few lines like the following to the end of the dump file:

```
--
-- Position to start replication from
--

CHANGE MASTER TO MASTER_LOG_FILE='bin.000846' ;
CHANGE MASTER TO MASTER_LOG_POS=427 ;
```

When the dump file is executed on the slave server, these last lines will record the name of the master's binary log file and the position in the log at the time of the backup, while the tables were locked. When replication is started, it will go to this log file and execute any SQL statements recorded starting from the position given. This is meant to ensure that any data changed while setting up the slave server isn't missed. To execute the dump file to set up the databases and data on the slave server, copy the dump file to the slave server, make sure MySQL is running, then enter something like the following on the slave:

```
mysql --user=root --password=my_pwd < /tmp/backup.sql
```

This will execute all of the SQL statements in the dump file, which will include the `CREATE` and `INSERT` statements. Once the backed-up databases are loaded onto the slave server, execute the following SQL statement while logged in as *root* on the slave:

```
START SLAVE;
```

After this statement is run, the slave will connect to the master and get the changes it missed since the backup. From there, it will stay current by continuously checking the binary log as outlined before.

Backups with Replication

With replication running, it's an easy task to make a backup of the data. You just need to temporarily stop the slave server from replicating by entering the following SQL statement while logging onto the slave server as *root* or a user with `SUPER` privileges:

```
STOP SLAVE;
```

The slave server knows the position where it left off in the binary log of the master server. So we can take our time making a backup of the replicated databases on the slave server. We can use any backup utility or method we prefer. When the backup is finished, we would enter the following SQL statement from the slave server as *root* to restart replication:

```
START SLAVE;
```

After entering this statement, there should be a flurry of activity on the slave as it executes the SQL statements that occurred while it was down. In a very short period of time it should be current.

Automatizing Backups

If replication and the backup process are working properly, we can write a simple shell script to stop replication, back up the data on the slave server, and start the slave again. Such a shell script would look something like this:

```
#!/bin/sh

date=`date +%Y%m%d`

mysqladmin --user=root --password=my_pwd stop-slave

mysqldump --user=root --password=my_pwd --lock-all-tables \
      --all-databases > /backups/mysql/backup-${date}.sql

mysqladmin --user=root --password=my_pwd start-slave
```

In this example, we're using the `mysqladmin` utility to stop and start replication on the slave. On the first line, you may have noticed that we're capturing the date using the system function `date` and putting it into a good format (e.g., `20050615`). This variable is used with `mysqldump` in the script for altering the name of the dump file each day. Of course, you can set the file path and the name of the dump file to your preferences. Notice that the `date` function and the formatting codes are enclosed in back-ticks (`` ` ``), not single quotes (`'`).

This is a simple script. You may want to write something more elaborate and allow for error checking. You probably would also want to compress the dump files to save space and write them to a removable media like a tape or CD. Once you have your script set up, test it. If it works, you can add it to your `crontab` or whatever scheduling utility you use on your server.

Conclusion

Replication is a useful administrative feature of MySQL. It's an excellent way to be assured of good regular backups of databases. There are more options and SQL statements available for replication than I was able to present here. I cover them individually in my book MySQL in a Nutshell. For active and large systems, you may want to set up more than one slave server for added protection of data. The configuration and concepts are the same for multiple slaves as it is for one slave. For extremely active and large databases, you might want to consider purchasing software like that offered by Emic. Their software costs a bit, but it does an excellent job of handing slave serves for backups and load balancing, especially.

In May 2005, O'Reilly Media, Inc., released MySQL in a Nutshell.

- Sample Chapter 6, "Date and Time Functions" (PDF format), is available free online.
- You can also look at the Table of Contents, the Index, and the full description of the book.
- For more information, or to order the book, click here.

Russell Dyer has worked full-time for several years as a free-lance writer of computer articles, primarily on MySQL.

Showing messages 1 through 15 of 15.

- **intial dump of master database**

  Hi there,

  Is there an alternative to dump the master database for replication to slave server? Our database is kinda big, and there are tables over 2GB in size.
  Thanks,

  David

- **Backup script**

  This page has helped me a great deal. I read all the documentation re: replication on the MySQL website, and still had some questions on how to perform this very task. This is a real nice script given how simple and effective it is.

  I think the coolest find in here for me was the mysqldump --master-data option when setting up the slave. I will also use this regularly when I backup the slave, that way if the master goes down and I need to reverse roles on the master and slave, having that option set should help get things back up to speed on the master a bit quicker. At least I hope!

- **Replication Problems**

  I have gone through your tutorial but have ended up with problems. When I do a mysql dump I don't get any lines for bin file or start position. Also when I upload the sql statement to the slave it remove all the slave user info I added. I would think you wouldn't want to export the mysql database. What is the point of adding a mysql replicant user that points to the master if your going to over write it with the master dump file. Also my servers have different names, and after dumping the file and adding it to the slave I can't access the database from my database software because the host names have changed.

  I would also assume that you would have to restart the mysql servers after altering the my.cnf info, am I correct?

  Thanks for any help you can give.

  - **Replication Problems**

It works really nicely the way the article describes if you create the replicant user without a specific host name, instead create it as `'replicant'@'%'`. If you change your root account to '`root'@'localhost`' you should be able to replicate the mysql database to the slave without causing problems when you overwrite it with the master dump file.

(for those that MAY not know that `@'%'` and `@'localhost'` means, `'%'` is for ANY host and '`localhost`' is for the logging in from the box itself only)

If you think you may ever have more than one slave, I'd recommend doing it this way, that way all slaves could become the master with no required changes to the user table. The only command needed to change master on the slaves would be 'CHANGE MASTER TO'.

This way if you have more than one slave you can promote any of the slaves to become a master and all remaining slaves can be pointed to replicate from the new master easily.

If you really want to have a different set of user accounts on the slave than on the master then I'd recommend using the `--ignore-table=mysql.xxxx` for each table you want to ignore replication.

Remember the whole point of this article is to set up replication for a working backup solution. So if you have different user accounts on the slave when you perform the backup, you'll need to have some way of restoring the user accounts associated with the master before you can call it a complete backup solution.

- **Replication Problems**

  Sorry for the delay in responding--I've been on the road and all, and dealing with hurricanes and what not. In case you haven't resolved this yet, though, here are some suggestions.

  Regarding the lines to include the binary file name and the starting position, the `--master-data` option with `mysqldump` should put these at the end of the dump file. If not, you can manually run them on the slave:

```
CHANGE MASTER TO MASTER_LOG_FILE = 'bin.000001';
CHANGE MASTER TO MASTER_LOG_POS = 4;
```

  You can get the correct values for your server by entering this statement on the master:

  SHOW MASTER STATUS;

  Regarding the user information, if there are already users on the slave--meaning it's not going to be used just for backups--and for security purposes, you probably should add the `--ignore-table=mysql.users` option to the mysqldump command when backing up the master. This will exclude the users table and thereby the passwords from the dump file. This should take care of the host name problem you mentioned related to the users, as well. If you utilize the other tables in the mysql database, since

they contain references to users and hostnames, you should add them to the mysqldump line. You will have to add a separate key/value pair for each: `--ignore-table=mysql.xxxx` for each. I should have included all of this in the article.

Let me make two more clarification points to my article: don't start the slave replicating until you have the data copied or else it will try to start replicating before you can get the dump file unloaded. The problem is that you need to have MySQL running on the slave so that you can add the dump file. **To get around this, after you have the slave configured initially, restart the mysqld daemon (or mysqld_safe) with the --skip-slave-start option. Actually, put that option in the my.cnf file to be safe. Just delete it after replication is running.**

There is another point I'd like to add now that I'm reading your comments and a couple others: if your master has been binary logging for quite a while before you tried setting up replication, unless you have use for the logs, you might want to start fresh by issuing a RESET MASTER statement on the master. This will delete all of the binary log files and it will commit any outstanding transactions (e.g., on InnoDB and BDB tables), so be sure that you want to do this. You might want to make an extra backup of the databases and the binary logs before you reset the master, by the way. After you reset the master, make your backup and copy it to the slave server while replication isn't running. Then start the slave with the START SLAVE statement. Starting fresh makes it easy to be assured of a good clean start. Check the SHOW PROCESSLIST; on both servers to see the states of each. Also, run SHOW SLAVE STATUS; on the slave to see if everything looks okay. It will list the last error number and message if replication failed after starting. Check the error logs for clues if there's a problem.

- **How can I change the original master to be the new slave when it comes back up**

  Hi,

  Thanks for the article. But can you please tell me How can I change the original master to be the new slave when it comes back up?

  My understanding is when the old master comes back up, it will configure as a "master" (because that is what its configuration said). But the old slave is now the "master", wouldn't this become a dual master scenario? Can you please tell me how to resolve that?

  Thank you.
  Sam

- **Partial replication**

  After reading the entire article I still have a question in mind. I'll explain you my setup and could you could tell me if it's something possible to do replication in these conditions?

  1- One MySQL server in production environment (with 4 databases in it)

2- One MySQL in development environment which contains more databases than the production server (7 instead of 4)

I would like to replicate my production server on my development server to be able to take a backup without shutting down my prod server but I still want to use my three other development databases. So does the two servers need to have the same number of databases in it or it doesn't matter?

From my point of view, it shouldn't work because it uses the master's binary log but I would like to know your opinion on that guys.

Thank you

- **What about when disaster strikes?**

  This was generally a good read, but as with many articles on backup I miss the really important part: if your database goes corrupt and you need to restore a backup from the replicating server over to the master server - what then?

  For instance there will undoubtedly be a log of SQL statements since the last backup that have executed and eventually lead to corruption. Most of these statements are probably valid up to the one that caused corruption (if it wasn't hardware failure). Some questions that arise:

  - After restoring the backed up database on the master server, can you execute a certain amount of these statements to try to determine which one caused the corruption?

  - And can you then run the logged statements up to that point to minimize data loss?

  - After doing that, how do you remove the statements that caused corruption and get the replicating slave back up and running properly?

  I think for an article like this to be really useful you need to provide the reader with information on how to do recovery and optionally/preferably a method for testing that your backups are sane (a step by step test-recovery of your databases for instance).
  - **What about when disaster strikes?**

    A way to manage that is use the opportunity of having a slave to perform backups on it frequently. This way, you will probably be able to have a backup of a working version of your DB.

- **bi-directional replication**

  I have two web servers. Both run a server management system based on MySQL databases. (So, they both have databases with the same name, but different content.) I currently back them up with a nightly mysql dump of each individual database, without taking down the servers. Could they (easily) be configured to use replication to back each other up, without the database name collisions being an issue?
  - **bi-directional replication**

2005-07-08 10:50:40

I don't think a master-master relationship is supported at this time. Maybe in new, future release. I read somewhere that it is possible, but unsupported by MySQL team.

- **Replica requirements**

  Thanks for this article - very timely as far as I'm concerned as I'm about to put a MySQL-based app live :-)

  What sort of performance does the replica server need relative to the master? It looks like the updates to the replica DB can happen asynchronously as another thread is polling the logfile, so perhaps it doesn't need to be an identical box to the master. But where does the point come that the backup is compromised because the replicant can't keep up?
  - **Replica requirements**

    If your slave is dedicated, it doesn't have to be very powerful, as it only receives Updates + Inserts.
    - **Replica requirements**

      Thanks for this article; I now have a replica of my production DB :-) However, I was using OS X (I suppose you'd call my server DAMP rather than LAMP) and had to change a couple of things. There's no /etc/my.cnf by default, you have to copy one from somewhere (there's a sample file at /usr/share/mysql/my-medium.cnf). Even so, once you've edited the files as described in this article you still explicitly have to run the 'CHANGE MASTER TO' SQL statement before you can 'START SLAVE'.
      - **Replica requirements**

        I'm responding pretty late, but for future reference for others, here's my response.

        You are correct. The lines I presented in the article to set the values for connecting to the master are only used one, when you first restart the slave server. They are copied to the master.info file, which is by default located in the data directory on the slave. Once the master.info file has the connection information stored in it, the server will ignore these configuration settings or options in the my.cnf file and even at the command-line when the server is restarted. The only way to change them after the master.info file is created is to edit the file--which is not a very good idea--or to use the CHANGE MASTER TO statement that you mentioned. Therefore, to adjust my article, you may as well not include them in the my.cnf file to begin with. Instead, enter them from the slave while logged into MySQL as root or the like:

        CHANGE MASTER TO MASTER_HOST = 'master_host';
        CHANGE MASTER TO MASTER_PORT = 3306;
        CHANGE MASTER TO MASTER_USER = 'replicant';
        CHANGE MASTER TO MASTER_PASSWORD = 'my_pwd';

You only need to execute these statements one time to set them since they're stored in the master.info file. They'll survive restarts. If you want to change something like the host name, though, then rerun just the statement with that particular parameter (i.e. MASTER_HOST).

Regarding MacIntosh, although my servers have Linux on them, my laptop now is an iBook G4. Yes, you're right: by default it is set up without the my.cnf file. You can copy some of the sample files like you did, or you can create a new my.cnf file with just the group headings (e.g., [mysqld]) and options that you need with a simple text editor like vi while logged in as root (i.e., su)from a terminal window.

How To Set Up Database Replication In MySQL

Version 1.1
Last edited: 01/14/2006

This tutorial describes how to set up database replication in MySQL. MySQL replication allows you to have an exact copy of a database from a master server on another server (slave), and all updates to the database on the master server are immediately replicated to the database on the slave server so that both databases are in sync. This is not a backup policy because an accidentally issued DELETE command will also be carried out on the slave; but replication can help protect against hardware failures though.

In this tutorial I will show how to replicate the database *exampledb* from the master with the IP address *192.168.0.100* to a slave. Both systems (master and slave) are running **Debian Sarge**; however, the configuration should apply to almost all distributions with little or no modification.

Both systems have MySQL installed, and the database *exampledb* with tables and data is already existing on the master, but not on the slave.

I want to say first that this is not the only way of setting up such a system. There are many ways of achieving this goal but this is the way I take. I do not issue any guarantee that this will work for you!

1 Configure The Master

First we have to edit */etc/mysql/my.cnf*. We have to enable networking for MySQL, and MySQL should listen on all IP addresses, therefore we comment out these lines (if existant):

```
#skip-networking
#bind-address           = 127.0.0.1
```

Furthermore we have to tell MySQL for which database it should write logs (these logs are used by the slave to see what has changed on the master), which log file it should use, and we have to specify that this MySQL server is the master. We want to replicate the database *exampledb*, so we put the following lines into */etc/mysql/my.cnf*:

```
log-bin = /var/log/mysql/mysql-bin.log
binlog-do-db=exampledb
server-id=1
```

Then we restart MySQL:

```
/etc/init.d/mysql restart
```

Then we log into the MySQL database as *root* and create a user with replication privileges:

```
mysql -u root -p
Enter password:
```

Now we are on the MySQL shell.

```
GRANT REPLICATION SLAVE, REPLICATION CLIENT ON *.* TO 'slave_user'@'%'
IDENTIFIED BY '<some_password>';
```

(Replace *<some_password>* with a real password!)

```
FLUSH PRIVILEGES;
```

Next (still on the MySQL shell) do this:

```
USE exampledb;
FLUSH TABLES WITH READ LOCK;
SHOW MASTER STATUS;
```

The last command will show something like this:

```
+---------------+----------+--------------+------------------+
| File          | Position | Binlog_do_db | Binlog_ignore_db |
+---------------+----------+--------------+------------------+
| mysql-bin.006 | 183      | exampledb    |                  |
+---------------+----------+--------------+------------------+
1 row in set (0.00 sec)
```

Write down this information, we will need it later on the slave!

Then leave the MySQL shell:

quit;

There are two possibilities to get the existing tables and data from *exampledb* from the master to the slave. The first one is to make a database dump, the second one is to use the *LOAD DATA FROM MASTER;* command on the slave. The latter has the disadvantage that the database on the master will be **locked** during this operation, so if you have a large database on a high-traffic production system, this is not what you want, and I recommend to follow the first method in this case. However, the latter method is very fast, so I will describe both here.

Method 1:

If you want to follow the first method, then do this:

mysqldump -u root -p<password> --opt exampledb > exampledb.sql
(Replace *<password>* with the real password for the MySQL user *root*!
Important: There is **no** space between *-p* and *<password>*!)

This will create an SQL dump of *exampledb* in the file *exampledb.sql*.
Transfer this file to your slave server!

Method 2:

If you want to go the *LOAD DATA FROM MASTER;*
way then there is nothing you must do right now.

Finally we have to unlock the tables in *exampledb*:

```
mysql -u root -p
Enter password:
UNLOCK TABLES;
quit;
```

Now the configuration on the master is finished. On to the slave...

---

2 Configure The Slave

On the slave we first have to create the database *exampledb*:

```
mysql -u root -p
Enter password:
CREATE DATABASE exampledb;
quit;
```

If you have made an SQL dump of *exampledb* on the master and have transferred it to the slave, then it is time now to import the SQL dump into our newly created *exampledb* on the slave:

mysql -u root -p<password> exampledb < /path/to/exampledb.sql
(Replace *<password>* with the real password for the MySQL user root!
Important: There is **no** space between *-p* and *<password>*!)
If you want to go the *LOAD DATA FROM MASTER;* way then there is nothing you must do right now.

Now we have to tell MySQL on the slave that it is the slave, that the master is *192.168.0.100*,

and that the master database to watch is *exampledb*.
Therefore we add the following lines to */etc/mysql/my.cnf*:

```
server-id=2
master-host=192.168.0.100
master-user=slave_user
master-password=secret
master-connect-retry=60
replicate-do-db=exampledb
```

Then we restart MySQL:

```
/etc/init.d/mysql restart
```

If you have not imported the master *exampledb* with the help of an SQL dump, but want to go the *LOAD DATA FROM MASTER;* way, then it is time for you now to get the data from the master *exampledb*:

```
mysql -u root -p
Enter password:
LOAD DATA FROM MASTER;
quit;
```

If you have phpMyAdmin installed on the slave you can now check if all tables/data from the master *exampledb* is also available on the slave *exampledb*.

Finally, we must do this:

```
mysql -u root -p
Enter password:
SLAVE STOP;
```

In the next command (still on the MySQL shell) you have to replace the values appropriately:

```
CHANGE MASTER TO MASTER_HOST='192.168.0.100', MASTER_USER='slave_user',
MASTER_PASSWORD='<some_password>', MASTER_LOG_FILE='mysql-bin.006',
MASTER_LOG_POS=183;
```

- *MASTER_HOST* is the IP address or hostname of the master (in this example it is *192.168.0.100*).
- *MASTER_USER* is the user we granted replication privileges on the master.
- *MASTER_PASSWORD* is the password of *MASTER_USER* on the master.
- *MASTER_LOG_FILE* is the file MySQL gave back when you ran *SHOW MASTER STATUS;* on the master.
- *MASTER_LOG_POS* is the position MySQL gave back when you ran *SHOW MASTER STATUS;* on the master.

Now all that is left to do is start the slave. Still on the MySQL shell we run

```
START SLAVE;
quit;
```

That's it! Now whenever *exampledb* is updated on the master, all changes will be replicated to *exampledb* on the slave. Test it!

mysqldump

The utility *mysqldump* provides a rather convenient way to dump existing data and table structures. Note that while *mysqldump* is not the most efficient method for creating backups (mysqlhotcopy is, described next), it does offer a convenient method for copying data and table structures which could then be used to repopulate another SQL server, that server not even necessarily being MySQL. The function *mysqldump* can be used to backup all databases, several databases, one database, or just certain tables within a given database. In this section, the syntax involved with each scenario is provided, followed with a few examples.

Using *mysqldump* to backup just one database:

```
%>mysqldump [options] db_name
```

Using *mysqldump* to backup several tables within a database:

```
%>mysqldump [options] db_name table1 table2 . . . tableN
```

Using *mysqldump* to backup several databases:

```
%>mysqldump [options] --databases [options] db_name1 db_name2 . . . db_nameN
```

Using *mysqldump* to backup all databases:

```
%>mysqldump [options] --all-databases [options]
```

The options can be viewed by executing the following command:

```
%>mysqldump --help
```

Examples:

Backing up both the structure and data found within the widgets database would be accomplished as follows:

```
%>mysqldump -u root -p --opt widgets
```

Alternatively, perhaps just a backup of the data is required. This is accomplished by including the option *--no-create-info*, which means no table creation data:

```
%>mysqldump -u root -p --no-create-info widgets
```

Another variation is just to backup the table structure. This is accomplished by including the option *--no-data*, which means no table data:

```
%>mysqldump -u root -p --no-data widgets
```

If you are planning on using *mysqldump* for reason of backing up data so it can be moved to another MySQL server, it is recommended that you use the option '--opt'. This will give you an optimized dump which will result in a faster read time when you later load it to another MySQL server.

While *mysqldump* provides a convenient method for backing up data, there is a second method which is both faster and more efficient. It is described in the next section.

mysqlhotcopy

The *mysqlhotcopy* utility is a perl script that uses several basic system and SQL commands to backup a database. More specifically, it will lock the tables, flush the tables, make a copy, and unlock the tables. Although it is the fastest method available for backing up a MySQL database, it is limited to backing up only those databases residing on the same machine as where it is executed.

The function *mysqlhotcopy* can be executed to backup one database, a number of databases, or only those databases matching a name specified by a regular expression. In this section, the syntax involved with each scenario is provided, followed with a few examples.

Using *mysqlhotcopy* to backup just one database:

```
%>mysqlhotcopy [options] db_name /path/to/new_directory
```

Using *mysqlhotcopy* to backup just several databases:

```
%>mysqlhotcopy [options] db_name_1 ... db_name_n /path/to/new_directory
```

Using *mysqlhotcopy* to backup only those tables within a given database that match a regular expression:

```
%>mysqlhotcopy [options] db_name./regex/
```

The options can be viewed by executing the following command:

```
%>mysqlhotcopy --help
```

Examples:
Experiment with *mysqlhotcopy* by backing up the widgets database to the directory path "`/usr/mysql/backups/`". Execute the following command:

```
%>mysqlhotcopy -u root -p widgets /usr/mysql/backups
```

As a second example, assume that the widgets database contains the tables: "`products2000`", "`products2001`", "`clientele2000`", and "`clientele2001`", with the four digits at the end of each name representing the year for which that data represents. The administrator wants to backup only those tables relative to the year "`2000`":

```
%>mysqlhotcopy -u root -p widgets./^.+('2000')$/ /usr/mysql/backups
```

In the above example, the regular expression `/^.+('2000')$/` tells *mysqlhotcopy* to only backup those tables ending with the string "`2000`".

Database Replication with MySQL

Description:
MySQL offers a way to replicate its databases to another server dynamically.
The server receiving the 'Write' SQL commands is called the master and it can send a copy of the SQL 'write' commands to its Slave(s) via replication feature.

Note:
I will use two terms in this article 'Write' SQL commands and 'Read' SQL commands.
A 'Write' SQL command is meant, any SQL command that does modifies a database, as opposed to a 'Read' SQL command which is one that only reads the database.

Principle of operation:
When the MySQL server receives an 'Write' SQL command, if the option `log-bin` is set in the configuration file then it saves it's 'Write' SQL commands in a binary format in the directory specified. After a slave has connected to the master, it receives automatically all the 'Write' SQL commands saved in the binary log files of the master that were not yet sent to the slave. A pointer to the last updated position in the master's binary log is kept in the slave, so that if the slave goes down for a period of time, as soon as it goes back up, it gets updated of all the '*new and not received yet*' 'Write' SQL commands form the master.

Cascading replication:
It is also possible to cascade a master to a slave which in turns will be the master to another slave. eg. A -> B -> C. For this to work the following extra option needs to be entered in the configuration file of the server that plays double roles:
(In our case: B server. It works as slave to A Server and as master to C server)

```
/etc/mysql/my.cnf
      ......
      [mysqld]
      .......
      log-slave-updates
```

Normal operation of replication:
For this to function it is recommended to use an extra replication user in the database '`mysql`', table '`user`' in the master and the slaves'. In our examples I'll create the user '`replicant`' for that purpose. This replication user will get only the rights that allows the replication to function.

Command to create the '`replicant`' user in all MySQL servers involved (master and slaves). Make sure thiese commands are using the same passwords for `replicant` user.

```
mysql -u root -p                (enter the mySQL root's password)
> GRANT REPLICATION SLAVE ON *.* TO replicant@"%" IDENTIFIED BY 'new-password';
> quit;
```

Configuring MySQL servers:

The MASTER SERVER:
in `/etc/mysql/my.cnf`

```
#------------- Replication MASTER settings ---------
[mysqld]
......
server-id = 67                  (unique number to the server)
log-bin = /var/log/mysql/bin.log
```

The SLAVE SERVER:
in `/etc/mysql/my.cnf`

```
#------------- Replication SLAVE settings ---------
[mysqld]
......
server-id = 44                  (unique number to the server)
master-host = 192.168.0.2
master-port = 3306
master-user = replicant
master-password = 'mysecret'

log-bin = /var/log/mysql/bin.log
log-bin-index = /var/log/mysql/log-bin.index
log-error = /var/log/mysql/error.log

relay-log = /var/log/mysql/relay.log
relay-log-info-file = /var/log/mysql/relay-log.info
relay-log-index = /var/log/mysql/relay-log.index
```

Managing the replication servers.
Starting from the master's databases:

- Stop the master and the slaves mysql servers
- Make a copy of the master's database directory `/var/lib/mysql` to the slaves on the same path.
- Give the ownership of the copied directory and its content to mysql user on all slaves.
- Delete the following files on master and slaves servers(if they exist):
  ```
  /var/lib/mysql/ib*
  /val/lib/mysql/master.info
  /var/log/mysql/log-bin.index
  /var/log/mysql/mysql-bin.*
  /var/log/mysql/relay*
  ```
- Start the master server first
- Start all the slaves servers one after the other.

- Take a look in `/var/log/mysql/error.log` of slaves to make sure they have connected correctly to the master.

Restarting a replication with existing databases:

- Stop the master and the slaves mysql servers
- Delete the following files on master and slaves servers(if they exist):
  ```
  /var/lib/mysql/ib*
  /val/lib/mysql/master.info
  ```

```
/var/log/mysql/log-bin.index
/var/log/mysql/mysql-bin.*
/var/log/mysql/relay*
```

- Start the master server first
- Start all the slaves servers one after the other.
- Take a look in `/var/log/mysql/error.log` of slaves to make sure they have connected correctly to the master.

Normal Operation:

- To ensure that the slave has the same data as the master (not more not less than the master), make sure that the slaves don't get written to. Otherwise the master will never get that data. It will stay written in the slave but unknown to the master, therefore creating a non-symmetric content of the databases.

- Any of the servers (master or slaves) can go down without disturbing the replication's integrity except that if the master goes down then the incoming Write commands won't be precessed and therefore be lost.

Redundancy set-up:

In some cases this behavior of the slave which takes write commands privately without the master knowing or disturbing replication, might be wanted, especially in the case of cascading replication. In this scenario the first master is simply a security buffer in case the real master fails (the one in the middle: B server).
eg. A -> B -> C.
In this scenario the 'Write' SQL commands are normally given to B server and if B fails, then given to A server(this is alternative controlled by the application).
'A' server would then accumulate the commands and would send them to B server as soon as B comes back ON. Normally B server would always replicate new changes to its slaves, except the from commands that have been received via replication from another master server. In this case, to allow also B server to replicate these 'write' SQL to its slaves, B server needs an extra option (**`log-slave-updates`**) in its configuration file(`/etc/mysql/my.cnf`):
eg.

```
[mysqld]
      ......
      log-slave-updates
```