MySQL Replication Tutorial

Mats Kindahl <mats@mysql.com>

Prerequisites

In order to not clash with an existing installation, we will not do a proper install of the MySQL server but rather run it from a dedicated tutorial directory. To manage the setups for the tutorial, we will use a number of utility scripts that you need to fetch. The scripts rely on having Perl installed and using at least version 5.6.0, but any later version should do.

Downloads

Material for this tutorial can be downloaded from the replication tutorial page on the MySQL Forge at `http://forge.mysql.com/wiki/Replication/Tutorial`

Create a tutorial directory

As a first step, you should create a directory where you can place all the files that we will use in this tutorial. In this document we will refer to that directory as `reptut/`. You can use another name if you like, in which case you should just substitute that name for the code samples in this document.

Get a MySQL 5.1 server

In order to be able to run the tests in this tutorial, it is necessary to have a MySQL 5.1 server. In order to avoid clashes with an existing installed server, it is necessary to have access to a binary distribution of the server. So, you have to download a binary distribution of the server from www.mysql.com into the the `reptut/` directory and unpack it there. You need to have a distribution without an installer, so taking either the Linux (non RPM packages) or the Windows without installer packages athttp://dev.mysql.com/downloads/mysql/5.1.html should work.

In most cases, the utility scripts will be smart enough to figure out what directory the server files are placed in, but you might have to give it a hint during the setup. Here is how my directory looks after I have downloaded and unpacked the binary distribution of the server.

```
mats@romeo:~/reptut$ ls -F
mysql-5.1.23-rc-linux-i686-glibc23/
mysql-5.1.23-rc-linux-i686-glibc23.tar.gz
```

This tutorial is developed for MySQL Server 5.1, and since some commands and syntax of some commands are different between 5.0 and 5.1 you might have to check the reference manual if you are going to work with this tutorial for 5.0. If you discover any discrepancies or that it works differently for 5.0, feel free to send me a comment and I will update the document with the information.

Get and unpack the utility programs

For this tutorial, there is a number of small utility programs that are used. We are using this package to avoid clashing with an existing installation on the computer, and this will also allow us to easily create and experiment with several servers running at the same time from the replication tutorial directory. Normally, a server is set up for replication by changing the existing `my.cnf` file for the server that is installed.

The utility programs are constructed to work from the `reptut/` directory (or whatever directory name you have picked), so you need to unpack them into that directory. Unpacking them will create a `scripts/` directory where the scripts are located. After having unpacked the utility package `reptut-utils.tar.gz` or `reptut-utils.zip`, you need to set up the basic configuration files and directories for the tutorial utility programs, which you do by calling the `server-adm` as follows:

```
$ ./scripts/server-adm setup
```

This will create a configuration file `server-config.pl` where all the data about the tutorial is kept as well as a directory for keeping temporary files. Among other things, it will try to find an unpacked server directory and ask you if you want to use it. Normally, you can just press return for this question, but you can enter another directory if you want. The script will also add some small scripts and files to the `reptut/` directory to make it easy to work with the server. Among other things, it will set up soft links to the `mysqld` and `mysql` programs in the `bin/` directory of the server.

Replication setup

Setting up a server as master

The steps that are needed to configure a server to be a master are:

1. Add `log-bin` and `server-id` options to `my.cnf` file
2. Start server and connect a client to the server
3. Add a replication user

4. Give the replication user `REPLICATION SLAVE` privileges

Configuration parameters needed for a master

In order for a server to work as a master, we need to have the binary log active and we need to have a server id assigned to the server. The server id is used to distinguish the servers from each others and should be assigned so that it is unique. Two servers with the same id will effectively be treated as if they are the same server. In other words, your configuration file for the master needs to have the following two lines (in boldface) added.

```
[mysqld]
server-id = 1
log-bin   = master-bin.log
...
```

Strictly speaking, the name for the log-bin option is not necessary, but it is usually a good idea to use explicit names and not rely on defaults. Also, it is usually a good idea to have a server id for all server, even if they are not currently masters. This makes it easy to make them a master once you decide that you need to.

It is also necessary to have a user on the master with `REPLICATION SLAVE` privileges that can be used by the slave to fetch changes. In reality any user can be used but it usually better to have a dedicated user for this role.

As the first step. we will create the configuration file for use when setting up the master using the tutorial utility script `add-server`.

```
$ ./scripts/server-adm add name=master roles=master
Creating file for master...done!
Bootstrapping server master...done!
```

This will create a MySQL configuration file for the server, bootstrap the server based on the implementation that is used, and enter the data about the server in the `server-config.pl` configuration file used for the tutorial utility scripts (and create that file if necessary). If you haven't run the setup previously, you will get questions about what server directory to use. In addition, it will add sections for the MySQL client as well, to make it easy to connect to the server.

To start the server, we open a separate window and start the server with the just generated defaults file, and it should start without problems:

```
$ ./mysqld --defaults-file=master.cnf
```

Creating a replication user and granting it replication rights

In order for a slave to be able to connect to a master and read any changes that are made to the database on the master server, it is necessary to have a user that have replication rights to the master. In theory, any user can be used, but it is usually practical to create a dedicated replication user and grant that user the replication rights. Recall that a user with replication rights can read *any* changes done to the master, which means that you have to trust both the machine as well as the network between you and the machine to avoid compromising security In order to secure the network between the master and the slave, it is possible to use an SSL connection.

We start by connecting to the server using the generated configuration file, and then proceed with creating a replication user and adding replication privileges to the account. When starting the MySQL client, the configuration file will set the prompt to the name of the server that you gave when adding it above using the `server-adm` script.

```
$ ./mysql --defaults-file=master.cnf -uroot

master> CREATE USER repl_user@localhost
Query OK, 0 rows affected (0.00 sec)

master> GRANT REPLICATION SLAVE ON *.*
     -> TO repl_user@localhost IDENTIFIED BY 'xyzzy';
Query OK, 0 rows affected (0.00 sec)
```

Setting up a server as slave

To set up a server to act as a slave, the following steps have to be done:

1. Add configuration options for the relay log to the configuration file
2. Direct the slave server to a master server
3. Start the slave
4. Test that replication works
5. Check what hosts are connected to a master using `SHOW SLAVE HOSTS`

Caveat. When adding a slave to an installation that have been running for a while, another approach has to be used to avoid the long time necessary for the slave to catch up with the master, but we will consider that case in the replication for read scale-out chapter below.

Adding configuration options

Although not strictly necessary, it is usually a good idea to configure the relay log names for

the slave. This is done by adding values for the `relay-log-index` and `relay-log` options to the configuration file:

```
[mysqld]
...
relay-log-index = slave-relay-bin.index
relay-log = slave-relay-bin
```

Directing slave server to master and starting replication

As a first step, we create a new server for the role of slave and start the server (in a separate window) in the following manner:

```
$ ./script/server-adm add name=slave roles=slave
Creating file for slave...done!
Bootstrapping server slave...done!

$ ./mysqld --defaults-file=slave.cnf
```

Now you will have a server running and we can direct it to replicate from the master you set up previously (make sure that you still have it running). In order to direct a slave to a master we need four pieces of information:

1. A host name or host IP address
2. A port number for the server (it defaults to 3306)
3. A user name for a user with replication privileges
4. A password for that user

The the second two pieces you have assigned yourself when setting up the server as a master, and the first two pieces you can get from the configuration we have set up using the `server-adm` utility script. You get get the information about a server by checking the configuration file for the server:

```
$ cat master.cnf
[mysqld]
...
port = 12000
socket = /tmp/master.sock
...
[mysql]
...
host = localhost
...
```

With this information, we can just start a client and issue a CHANGE MASTER TO command

to direct the slave at the master and then start the slave.

```
slave> CHANGE MASTER TO
    ->    MASTER_HOST = 'localhost',
    ->    MASTER_PORT = 12000,
    ->    MASTER_USER = 'repl_user',
    ->    MASTER_PASSWORD = 'xyzzy';

slave> START SLAVE;
```

Testing replication

Everything is now set up so that you test if replication work. Connect a client to the master and make a change there to see that everything works. In this example, we will just create a table, insert something into it, and see that it works as expected.

```
$ ./mysql --defaults-file=master.cnf
master> CREATE TABLE tbl (a CHAR(20));
Query OK, 0 rows affected (0.57 sec)

master> INSERT INTO tbl VALUES ('Yeah! Replication!');
Query OK, 1 row affected (0.00 sec)

master> quit

$ ./mysql --defaults=file=slave.cnf

slave> SHOW TABLES;
+----------------+
| Tables_in_test |
+----------------+
| tbl            |
+----------------+
1 row in set (0.00 sec)

slave> SELECT * FROM tbl;
+--------------------+
| a                  |
+--------------------+
| Yeah! Replication! |
+--------------------+
1 row in set (0.00 sec)

slave> quit
```

The binary log

Now that we have set up replication and made it to work, we are ready to take a closer look at

the pieces that make up replication. In this section, we will go over how changes are propagated to the slave and investigate the files that are used to store information about replication progress and configuration.

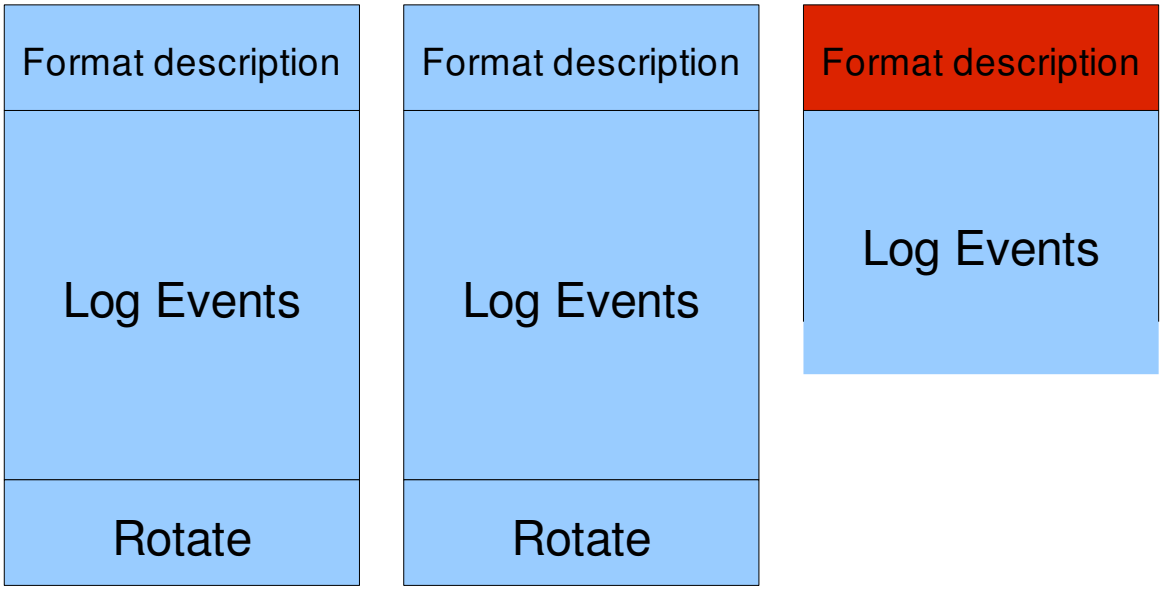
Changes done on the master is written to a *binary log*, which is then sent piece-by-piece to the slave. In this part, we will investigate the contents of the binary log, how to maintain and work with the binary logs, and demonstrate how replication is done using the binary log.

1. Get a list of the binary logs on the master
2. Investigate contents of the binary log
3. What is the difference between the binary log formats?

Working with the binary log files

To handle the binary log, there are several binary log *files* that together form the history of all changes ever done to the master. Each binary log file consists of a sequence of event, where the first event is a format description log event and the last event is a rotate event if it is a non-active binary log file. If the binary log file active, there is no rotate event written last (yet) and the header event indicates that this binary log file is not yet closed.

Whenever the binary logs are rotated, a rotate event is written last in the binary log, the binary log is marked as complete in the header event, and a new binary log file is created and a format description log event is written to it.



What binary log files are there?

To see what binary log files that are available, the `SHOW BINARY LOGS` command can be used. This command requires SUPER privileges, which means that you have to log in using the root account.

```
$ ./mysql --defaults-file=master.cnf -uroot

master> SHOW BINARY LOGS;
+-------------------+-----------+
| Log_name          | File_size |
+-------------------+-----------+
| master-bin.000001 |       660 |
| master-bin.000002 |       574 |
| master-bin.000003 |       106 |
+-------------------+-----------+
2 rows in set (0.00 sec)
```

Purging binary logs

As time passes, there will be more and more binary logs accumulating, most of which you will not need. These logs can be purged to save some disk. Binary logs can be purged either manually or automatically, and if they are purged manually they can be purged either by number or by date. The following is an example of purging all binary logs up to, but not including, `master-bin.000002`:

```
master> PURGE BINARY LOGS TO 'master-bin.000002';
Query OK, 0 rows affected (0.60 sec)

mysql> SHOW BINARY LOGS;
+-------------------+-----------+
| Log_name          | File_size |
+-------------------+-----------+
| master-bin.000002 |       574 |
| master-bin.000003 |       106 |
+-------------------+-----------+
2 rows in set (0.00 sec)
```

Also, it is possible to purge all binary logs before a certain date using with the same command. For example, to purge all binary log files except the current one, the following command can be used:

```
mysql> PURGE BINARY LOGS BEFORE NOW();
Query OK, 0 rows affected (0.46 sec)

mysql> SHOW BINARY LOGS;
+-------------------+-----------+
```

```
| Log_name          | File_size |
+-------------------+-----------+
| master-bin.000003 |       106 |
+-------------------+-----------+
1 row in set (0.00 sec)
```

Caveat. It is safe to purge binary log files that are active (i.e., the log file that is currently being written to). If a binary log is active, purging it will result in an error message. However, be aware that the binary logs represent the full change history of the master and are used for point-in-time recovery and when adding new slaves to a master. For that reason, it can be wise to make a backup the the binary logs before purging them. You should at least keep binary logs around since the last backup, in order to be able to do a point-in-time recovery.

A look at the contents of the binary log

In this section we will take a look at the binary log, see what different events exist in the binary log, and go through what purpose they have. This will not be an exhaustive walk-through of all events, but rather just a brief introduction to the workings of the binary log. To get detailed knowledge, it is necessary to study the reference manual and the code of the server. We will in this part also assume that we are working with statement-based replication and leave any special issues regarding row-based replication to later.

Browsing events in the binary log

In order to see what log events there are in the binary log, the `SHOW BINLOG EVENTS` command can be used.

```
master> SHOW BINLOG EVENTS;
```

There are six fields in the output:

`Log_name`	The binary log file name for this event
`Pos`	The binary log position of the event
`Event_type`	The event type, for example, `Query_log_event`
`Server_id`	The original server id of the event, I.e., the server id of the server that created this event originally
`End_log_pos`	The end log position
`Info`	Information about the event. For query log events, it is the query that was executed

A closer look at what goes into the binary log

When executing a statement in the server that changes data, it will be written into the binary log as a Query log event, which is then transported to the slave and executed there. In order to execute the statement in the correct database, the server adds a use statement before the actual statement. The database used is the *current database*, which is the database that the statement was executed in. A typical output can look as follows.

```
master> show binlog events\G
*************************** 1. row ***************************
   Log_name: master1-bin.000001
        Pos: 4
 Event_type: Format_desc
  Server_id: 10
End_log_pos: 106
       Info: Server ver: 5.1.23-rc-log, Binlog ver: 4
       Info: Server ver: 5.1.23-rc-log, Binlog ver: 4
*************************** 2. row ***************************
   Log_name: master1-bin.000001
        Pos: 106
Event_type: Query
  Server_id: 10
End_log_pos: 197
       Info: use `test`; create table t1 (a char(40))
*************************** 3. row ***************************
   Log_name: master1-bin.000001
        Pos: 197
 Event_type: Query
  Server_id: 10
End_log_pos: 301
       Info: use `test`; insert into t1 values ('Stuck In A Loop')
```

However, since the slave thread is executing all statements using a single thread at the slave, there are situations where it is necessary to know the context of the statement The typical case where the context is provided as well is when you are using a user variable inside a statement. In this case, the contents of the user variable is passed just before the statement is written to the binary log.

```
master> SET @TITLE = 'Post Post-Modern Man';
Query OK, 0 rows affected (0.00 sec)

master> INSERT INTO t1 VALUES(@TITLE);
Query OK, 1 row affected (0.00 sec)

master> SHOW BINLOG EVENTS FROM 301\G
*************************** 1. row ***************************
   Log_name: master1-bin.000001
        Pos: 301
 Event_type: User var
  Server_id: 10
```

```
End_log_pos: 359
       Info: @`title`=_ascii
0x506F737420506F73742D4D6F6465726E204D616E COLLATE
ascii_general_ci
*************************** 2. row ***************************
   Log_name: master1-bin.000001
        Pos: 359
 Event_type: Query
  Server_id: 10
End_log_pos: 451
       Info: use `test`; insert into t1 values(@title)
2 rows in set (0.00 sec)
```

Using `mysqlbinlog`

Working from within the server has a few drawbacks, such as that it is necessary to have a server running. Sometimes it is necessary to investigate the contents of the binary log and extract parts of the binary log to reconstruct a database. One of the more important tools for that is the `mysqlbinlog` tool. By default, mysqlbinlog will print the contents of a binary log as a text consisting of comments and SQL statements, which can look as follows.

```
$ ./mysqlbinlog master/log/master-bin.000001
/*!40019 SET @@session.max_insert_delayed_threads=0*/;
/*!50003 SET @OLD_COMPLETION_TYPE=@@COMPLETION_TYPE,COMPLETION_TYPE=0*/;
DELIMITER /*!*/;
# at 4
#080412 13:28:55 server id 10  end_log_pos 106  Start: binlog v 4,
#                server v 5.1.23-rc-log
#                created 080412 13:28:55 at startup
ROLLBACK/*!*/;
# at 106
#080412 13:33:24 server id 11  end_log_pos 192  Query   thread_id=1
#                exec_time=55    error_code=0
use test/*!*/;
SET TIMESTAMP=1208032404/*!*/;
SET @@session.foreign_key_checks=1, @@session.sql_auto_is_null=1,
@@session.unique_checks=1/*!*/;
SET @@session.sql_mode=0/*!*/;
/*!\C latin1 *//*!*/;
SET @@session.character_set_client=8,
    @@session.collation_connection=8,
    @@session.collation_server=8/*!*/;
create table t1 (a int)/*!*/;
```

As you can see, each of the actual statements are preceded by a set of SQL statements that make up the context for the execution of the statement. The intention is that you can use `mysqlbinlog` to extract information from a binary log, and then feed the output into a running server using the `mysql` client program.

Replication files on slave

We are now ready to start look closer at how replication works and what files are used to handle replication.

The most important files on the slave are the relay log files. They serve as a cache for the statements that are executed on the master. The relay log is written by the slave I/O thread, and read by the slave SQL thread. To keep track of the reading and application of events on the slave, there is a number of files containing information about the progress of replication; the files are `master.info`, and `relay-log.info`.

The `master.info` file

The `master.info` file is used to hold information about the master being replicated from and how much of the master binary log that has been replicated. There is one item for each row, and the file looks as follows:

```
15              # Lines in file
master-bin.001 # Log name
802             # Log pos
127.0.0.1       # Host
root            # User
xyzzy           # Password
9306            # Port
1               # Master connect retry
0               # Master SSL
                # Trusted Certification Authorities SSL
                # Path to directory of certificates
                # SSL certificate for this connection
                # List of allowed ciphers for this connection
                # Name of key file for connection
0               # Verify server certificate (since 5.1.16)
```

The relay-log.`info` file

The `relay-log.info` file is used to keep track of how much of the binary log has been applied. In this file, the figures give what position in the relay log that has been applied, and what position and the master binary log that it corresponds to.

```
slave-relay-bin.000001  # Relay log name
856                     # Relay log pos
master-bin.00001        # Master log name
802                     # Master log pos
```

Basic replication use scenarios

In this section we will go through a few basic scenarios for using replication.

1. Redundancy, or high-availability
2. Load balancing reads
3. Offline processing to avoid stopping the master

Using a slave for taking a backup

In order to get a backup of a server without stopping it, you can either use some of the online backup tools or set up a dedicated slave that get all changes from the master. You can then stop the slave, take an offline backup of it (even doing a physical backup by just copying the database directories), and then start the slave again.

Using a slave for reporting or analysis

For similar reasons, a slave can be used for offline processing of data to, for example, generate reports based on data in the database.

Replication for read scale-out

Replication for read scale-out is focused on having many slaves replicating data from a master and various ways for doing that efficiently. The goal of this is to relieve the master from read queries when load on the master becomes to high. To do this, slaves are added and clients should be redirected to the slave when doing reads, but writes still have to go to the master.

Adding new slaves

Assuming that the master have been running for a long time, the number of changes in the binary log can be considerable. Even though we can connect to the master and read the binary log from the beginning, it would take a long time for the slave to catch up with the master. Instead, we clone an existing slave by taking a backup of the slave, restore the backup on the new slave, and then start the slave replicating from the position that corresponds to the position in the binary log that the backup corresponds to.

There is a common way to take a backup of a slave that only works when the load on the master is low, but which fails when the load on the master is high. I have outlined that approach in the appendix, and also show why it does not work. We will here instead

concentrate on cloning a slave in such a manner that we do not have to bring the master down, and instead rely on having an existing slave that we can clone.

How to clone a slave

In order to clone a slave, we have to decide on a point in the master binary log where we want to stop the slave and use the `START SLAVE UNTIL` command to have the slave stop *exactly* at a specified position. When we have a master position, and we have stopped the slave at exactly this position, we have a snapshot of the master corresponding to a known position in the binary log. That way, we know where to start the new slave, and we can also take a backup of the existing slave.

The easiest way of getting a position in the master binary log where we can stop the slave is to use the `FLUSH TABLES WITH READ LOCK` on the master to flush all tables and add a read lock to the tables of the database, get the master position, and then release the locks. This will mean that the master is stopped just a brief moment to allow us to get a proper position to use as synchronization point. After we have that position, we can use the `START SLAVE UNTIL` to have the slave stop at the right position.

The steps that has to be done to get a backup and a binlog position for what the backup corresponds to are:

1. Stop slave
2. Flush tables on master with a read lock
3. Get master position
4. Unlock tables to let the master start running again
5. Start the slave to run until the binlog position from point 3
6. Wait for the slave to stop
7. Take a backup of the slave and take a note of the binary log file and position that it corresponds to
8. Start the slave again

In other words, the steps that need to be done are illustrated in the following code sample.

```
slave> STOP SLAVE;
Query OK, 0 rows affected, 1 warning (0.00 sec)

master> FLUSH TABLES WITH READ LOCK;
Query OK, 0 rows affected (1.31 sec)

master> SHOW MASTER STATUS;
+------------------+----------+--------------+------------------+
| File             | Position | Binlog_Do_DB | Binlog_Ignore_DB |
+------------------+----------+--------------+------------------+
| master-bin.000003 |      742 |              |                  |
+------------------+----------+--------------+------------------+
```

```
master> UNLOCK TABLES;
Query OK, 0 rows affected (0.00 sec)

slave> START SLAVE UNTIL
    ->     MASTER_LOG_FILE='master-bin.000003',
    ->     MASTER_LOG_POS=742;
Query OK, 0 rows affected (0.01 sec)

slave> DO MASTER_POS_WAIT('master-bin.000003', 742);
Query OK, 0 rows affected (2.17 sec)

$ cat >slave/data/binlog_pos.dat
master-bin.000003
742

$ tar cxf slave-backup.tar.gz slave/data

slave> START SLAVE;
Query OK, 0 rows affected (0.01 sec)
```

Since this is quite intricate, the easiest is to automate it. I have added a sample Perl script `take-backup.pl` that do this to the replication tutorial software package. You have to modify the script and change the backup code to use the form of backup that you want to use. In the script, the database is simply archived and compressed using `tar(1)`.

Creating the new slave

Now that we have a backup to use when bootstrapping the new slave, we add another slave to the set of slaves using the server-adm script that we used in the previous section and start the server.

```
$ ./scripts/server-adm add name=slave2 role=slave

$ ./mysqld --defaults-file=slave2.cnf
```

Now that we have the slave running, we copy the backup into the data directory of the new slave and tell the slave to start from the binary log position that we took a note of when taking the backup. We can do that with the new slave running, since the tables will automatically be discovered.

```
$ tar zxf slave-backup.tar.gz slave2/data

$ cat slave2/data/binlog_pos.dat
master-bin.000003
742

$ ./mysql --defaults-file=slave1.cnf -uroot
slave2> CHANGE MASTER TO
     ->    MASTER_HOST = 127.0.0.1,
     ->    MASTER_PORT = 12000,
```

```
    ->    MASTER_USER = 'repl_user',
    ->    MASTER_PASSWORD = 'xyzzy',
    ->    MASTER_LOG_FILE = 'master-bin.000003',
    ->    MASTER_LOG_POS = 742;

slave2> START SLAVE;
```

In order to simplify that job,I wrote a small script `load-slave` to load the slave data from a tar file produced by the `take-backup` script.

```
$ ./scripts/load-slave slave2 slave-backup.tar.gz
```

Comparing progress of slaves

In order to see the progress of a set of slave servers, you can connect to each server in turn and check the status of replication using `SHOW SLAVE STATUS`. In order to do that, you need to have a user on each slave with `REPLICATION CLIENT` privileges. Following is a small script that can be used to query the status of a set of servers:

Replication for high-availability

The other use for replication is to implement high-availability by having two master replication to each others. We will in this section demonstrate how to set up such a replication scenario and show the problems with it and outline how it can be handled.

Dual masters

In order for a pair to work as dual master, they have to server both as master and as slave. They serve as master regarding any changes that comes to it directly from clients, but they also serve as slave in the sense that changes are replicated from another master. By default, when replicating from a master, changes that come from the master is *not* logged to the binary log unless the `log-slave-updates` option is given to the server. If the option is not supplied, changes cannot be replicated further. This means that in order for a server that acts both in the role as slave and as master, it is necessary to add this option to the configuration file.

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

Setting up dual masters

We start by setting up two masters using the `server-adm` script. If we assign both the roles

`master` and `slave` to the master, the `server-adm` script will add the `log-slave-updates` to the configuration file. Since we are starting with a fresh setup, we will also reset the master to remove all binary log files before setting up the replication. Here is what needs to be done to set up one master to act as a pair in a dual master setup.

```
$ ./script/server-adm add name=master1 roles=master,slave

$ ./mysql --defaults-file=master1.cnf -uroot

master1> CREATE USER repl_user@localhost;

master1> GRANT REPLICATION SLAVE ON *.*
      -> TO repl_user@localhost IDENTIFIED BY 'xyzzy';

master1> RESET MASTER;
```

After the masters are set up, we are ready to start the replication. We do that by issuing a `CHANGE MASTER TO` command as previously on both master/slaves and then start the slave threads. Note that we have to configure both the masters as above before we start the replication, but once the configuration is set up correctly, we can start the masters in any order.

```
master1> CHANGE MASTER TO
      ->    MASTER_HOST = 127.0.0.1,
      ->    MASTER_PORT = 12000,
      ->    MASTER_USER = 'repl_user',
      ->    MASTER_PASSWORD = 'xyzzy';

master1> START SLAVE;
```

After we have configured the masters, we can connect a slave and direct it to either of the masters, and then start it. Changes done to either master will then propagate to the slave.

Testing replication

We are now ready to test replication. Since we want to check that changes done to either master actually replicate to the other master, we start by resetting the previous slave that we set up and reconnect it again to the same master.

```
$ ./mysql --defaults-file=slave.cnf -uroot

slave> STOP SLAVE;
Query OK, 0 rows affected (0.01 sec)

slave> RESET SLAVE;
Query OK, 0 rows affected (0.00 sec)
```

```
slave> START SLAVE;
Query OK, 0 rows affected (0.00 sec)
```

After that, we connect to the each master in turn, make some changes, and see that all changes propagate to the slave.

```
$ ./mysql --defaults-file=master.cnf -uroot

master> CREATE TABLE t1 (a INT);
Query OK, 0 rows affected (0.01 sec)

master> INSERT INTO t1 VALUES (1),(2);
Query OK, 2 rows affected (0.01 sec)

$ ./mysql --defaults-file=master1.cnf -uroot

master1> INSERT INTO t1 VALUES (3),(4);
Query OK, 2 rows affected (0.01 sec)

master1> SELECT * FROM t1;
+---+
| a |
+---+
| 1 |
| 2 |
| 3 |
| 4 |
+---+
4 rows in set (0.00 sec)

$ ./mysql --defaults-file=slave.cnf -uroot

slave> SHOW TABLES;
+----------------+
| Tables_in_test |
+----------------+
| t1             |
+----------------+
1 row in set (0.00 sec)

slave> SELECT * FROM t1;
+---+
| a |
+---+
| 1 |
| 2 |
| 3 |
| 4 |
+---+
4 rows in set (0.00 sec)
```

Appendix. A commonly used but incorrect way to stop the slave

A common, but unsafe, way to stop the slave is based on the way the test system works, so we will check closer how the test system tries to get a synchronization point and see why that does not work for us. Suppose that we have a master and slave set up according to how was previously demonstrated and that the slave is running and up to date. Now, we assume that the master is under heavy use, and there are writes going on all the time.

If you are familiar with the test system, there is a test command by the name `save_master_pos`. The intention is that this command saves the binary log position of the last command, and you can then use the `sync_with_master` command to make the test wait until the slave has caught up with the master, and then continue working. So, for example, the following short test script intended to show the same values on the master and the slave for table `tbl`.

```
connection master;
INSERT INTO tbl VALUES ('Careful with that axe, Eugene!');
SELECT * FROM tbl;
save_master_pos;
connection slave;
sync_with_master;
SELECT * FROM tbl;
```

The `save_master_pos` is implemented in `mysqltest` by using `SHOW MASTER STATUS` to get the last written position in the binary log, and the `sync_with_master` is implemented using the `master_pos_wait()` function, supplied with the information from the `SHOW MASTER STATUS` command. In other words, the statements that the test system executes as-if the following sequence of commands were executed:

```
master> INSERT INTO tbl VALUES ('Careful with that axe, Eugene!');
Query OK, 1 row affected (0.00 sec)

master> SELECT * FROM tbl;
+--------------------------------+
| str                            |
+--------------------------------+
| Careful with that axe, Eugene! |
+--------------------------------+
1 row in set (0.00 sec)

master> SHOW MASTER STATUS;
+-------------------+----------+--------------+------------------+
| File              | Position | Binlog_Do_DB | Binlog_Ignore_DB |
+-------------------+----------+--------------+------------------+
| master-bin.000001 |      440 |              |                  |
+-------------------+----------+--------------+------------------+
1 row in set (0.00 sec)
```

```
slave> DO MASTER_POS_WAIT('master-bin.000001', 440);
Query OK, 0 rows affected (0.01 sec)

slave> SELECT * FROM tbl;
+-------------------------------+
| str                           |
+-------------------------------+
| Careful with that axe, Eugene! |
+-------------------------------+
1 row in set (0.00 sec)
```

So, can't we do something like that? In other words, wouldn't something along the following lines work?

```
master> SHOW MASTER STATUS;

slave> DO MASTER_POS_WAIT(...);

slave> STOP SLAVE;

$ tar zcf slave-backup.tar.gz slave/data

slave> START SLAVE;
```

Unfortunately not. The problem is that the slave keeps running after the wait, which means that there potentially can be several inserts done into the binary log between that statement and stopping the slave. This in turn means that we cannot trust the value of the `Exec_Master_Log_Pos` nor the `Master_Log_File` field of `SHOW SLAVE STATUS`.