

# MySQL Replication

---

# MySQL Replication

## Abstract

This is the MySQL Replication extract from the MySQL 5.0 Reference Manual.

Document generated on: 2012-09-05 (revision: 32068)

---

# Table of Contents

Preface and Legal Notices .....	v
1. Replication .....	1
2. Replication Configuration .....	3
How to Set Up Replication .....	4
Setting the Replication Master Configuration .....	5
Setting the Replication Slave Configuration .....	5
Creating a User for Replication .....	6
Obtaining the Replication Master Binary Log Coordinates .....	6
Creating a Data Snapshot Using <code>mysqldump</code> .....	7
Creating a Data Snapshot Using Raw Data Files .....	8
Setting Up Replication with New Master and Slaves .....	9
Setting Up Replication with Existing Data .....	10
Introducing Additional Slaves to an Existing Replication Environment .....	12
Setting the Master Configuration on the Slave .....	13
Replication and Binary Logging Options and Variables .....	13
Replication and Binary Logging Option and Variable Reference .....	14
Replication Master Options and Variables .....	17
Replication Slave Options and Variables .....	20
Binary Log Options and Variables .....	39
Common Replication Administration Tasks .....	44
Checking Replication Status .....	45
Pausing Replication on the Slave .....	46
3. Replication Solutions .....	49
Using Replication for Backups .....	49
Backing Up a Slave Using <code>mysqldump</code> .....	50
Backing Up Raw Data from a Slave .....	50
Using Replication with Different Master and Slave Storage Engines .....	51
Using Replication for Scale-Out .....	52
Replicating Different Databases to Different Slaves .....	54
Improving Replication Performance .....	55
Switching Masters During Failover .....	56
Setting Up Replication Using SSL .....	59
4. Replication Notes and Tips .....	61
Replication Features and Issues .....	61
Replication and <code>AUTO_INCREMENT</code> .....	62
Replication and Character Sets .....	63
Replication of <code>CREATE TABLE ... SELECT</code> Statements .....	63
Replication of <code>DROP ... IF EXISTS</code> Statements .....	63
Replication and <code>DIRECTORY</code> Table Options .....	64
Replication and Floating-Point Values .....	64
Replication and <code>FLUSH</code> .....	64
Replication and System Functions .....	64
Replication and <code>LIMIT</code> .....	66
Replication and <code>LOAD</code> Operations .....	66
Replication and the Slow Query Log .....	66
Replication and <code>REPAIR TABLE</code> .....	66
Replication and Master or Slave Shutdowns .....	67
Replication and <code>MEMORY</code> Tables .....	67
Replication and Temporary Tables .....	68
Replication of the <code>mysql</code> System Database .....	68
Replication and User Privileges .....	69

Replication and the Query Optimizer .....	69
Replication and Reserved Words .....	69
Slave Errors During Replication .....	69
Replication and Server SQL Mode .....	70
Replication Retries and Timeouts .....	70
Replication and <code>TIMESTAMP</code> .....	70
Replication and Time Zones .....	70
Replication and Transactions .....	71
Replication and Triggers .....	71
Replication and Views .....	72
Replication and Variables .....	72
Replication Compatibility Between MySQL Versions .....	73
Upgrading a Replication Setup .....	74
Troubleshooting Replication .....	75
How to Report Replication Bugs or Problems .....	76
5. Replication Implementation .....	79
Replication Implementation Details .....	79
Replication Relay and Status Logs .....	81
The Slave Relay Log .....	81
Slave Status Logs .....	82
How Servers Evaluate Replication Filtering Rules .....	84
Evaluation of Database-Level Replication and Binary Logging Options .....	84
Evaluation of Table-Level Replication Options .....	86
Replication Rule Application .....	90

---

# Preface and Legal Notices

This is the MySQL Replication extract from the MySQL 5.0 Reference Manual.

## Legal Notices

Copyright © 1997, 2012, Oracle and/or its affiliates. All rights reserved.

This software and related documentation are provided under a license agreement containing restrictions on use and disclosure and are protected by intellectual property laws. Except as expressly permitted in your license agreement or allowed by law, you may not use, copy, reproduce, translate, broadcast, modify, license, transmit, distribute, exhibit, perform, publish, or display any part, in any form, or by any means. Reverse engineering, disassembly, or decompilation of this software, unless required by law for interoperability, is prohibited.

The information contained herein is subject to change without notice and is not warranted to be error-free. If you find any errors, please report them to us in writing.

If this software or related documentation is delivered to the U.S. Government or anyone licensing it on behalf of the U.S. Government, the following notice is applicable:

U.S. GOVERNMENT RIGHTS Programs, software, databases, and related documentation and technical data delivered to U.S. Government customers are "commercial computer software" or "commercial technical data" pursuant to the applicable Federal Acquisition Regulation and agency-specific supplemental regulations. As such, the use, duplication, disclosure, modification, and adaptation shall be subject to the restrictions and license terms set forth in the applicable Government contract, and, to the extent applicable by the terms of the Government contract, the additional rights set forth in FAR 52.227-19, Commercial Computer Software License (December 2007). Oracle USA, Inc., 500 Oracle Parkway, Redwood City, CA 94065.

This software is developed for general use in a variety of information management applications. It is not developed or intended for use in any inherently dangerous applications, including applications which may create a risk of personal injury. If you use this software in dangerous applications, then you shall be responsible to take all appropriate fail-safe, backup, redundancy, and other measures to ensure the safe use of this software. Oracle Corporation and its affiliates disclaim any liability for any damages caused by use of this software in dangerous applications.

Oracle is a registered trademark of Oracle Corporation and/or its affiliates. MySQL is a trademark of Oracle Corporation and/or its affiliates, and shall not be used without Oracle's express written authorization. Other names may be trademarks of their respective owners.

This software and documentation may provide access to or information on content, products, and services from third parties. Oracle Corporation and its affiliates are not responsible for and expressly disclaim all warranties of any kind with respect to third-party content, products, and services. Oracle Corporation and its affiliates will not be responsible for any loss, costs, or damages incurred due to your access to or use of third-party content, products, or services.

This document in any form, software or printed matter, contains proprietary information that is the exclusive property of Oracle. Your access to and use of this material is subject to the terms and conditions of your Oracle Software License and Service Agreement, which has been executed and with which you agree to comply. This document and information contained herein may not be disclosed, copied, reproduced, or distributed to anyone outside Oracle without prior written consent of Oracle or as specifically provided below. This document is not part of your license agreement nor can it be incorporated into any contractual agreement with Oracle or its subsidiaries or affiliates.

This documentation is NOT distributed under a GPL license. Use of this documentation is subject to the following terms:

You may create a printed copy of this documentation solely for your own personal use. Conversion to other formats is allowed as long as the actual content is not altered or edited in any way. You shall not publish or distribute this documentation in any form or on any media, except if you distribute the documentation in a manner similar to how Oracle disseminates it (that is, electronically for download on a Web site with the software) or on a CD-ROM or similar medium, provided however that the documentation is disseminated together with the software on the same medium. Any other use, such as any dissemination of printed copies or use of this documentation, in whole or in part, in another publication, requires the prior written consent from an authorized representative of Oracle. Oracle and/or its affiliates reserve any and all rights to this documentation not expressly granted above.

For more information on the terms of this license, or for details on how the MySQL documentation is built and produced, please visit [MySQL Contact & Questions](#).

For additional licensing information, including licenses for third-party libraries used by MySQL products, see [Preface and Legal Notices](#).

For help with using MySQL, please visit either the [MySQL Forums](#) or [MySQL Mailing Lists](#) where you can discuss your issues with other MySQL users.

For additional documentation on MySQL products, including translations of the documentation into other languages, and downloadable versions in variety of formats, including HTML and PDF formats, see the [MySQL Documentation Library](#).

---

# Chapter 1. Replication

Replication enables data from one MySQL database server (the master) to be replicated to one or more MySQL database servers (the slaves). Replication is asynchronous - slaves need not be connected permanently to receive updates from the master. This means that updates can occur over long-distance connections and even over temporary or intermittent connections such as a dial-up service. Depending on the configuration, you can replicate all databases, selected databases, or even selected tables within a database.

For answers to some questions often asked by those who are new to MySQL Replication, see [MySQL 5.0 FAQ: Replication](#).

The target uses for replication in MySQL include:

- Scale-out solutions - spreading the load among multiple slaves to improve performance. In this environment, all writes and updates must take place on the master server. Reads, however, may take place on one or more slaves. This model can improve the performance of writes (since the master is dedicated to updates), while dramatically increasing read speed across an increasing number of slaves.
- Data security - because data is replicated to the slave, and the slave can pause the replication process, it is possible to run backup services on the slave without corrupting the corresponding master data.
- Analytics - live data can be created on the master, while the analysis of the information can take place on the slave without affecting the performance of the master.
- Long-distance data distribution - if a branch office would like to work with a copy of your main data, you can use replication to create a local copy of the data for their use without requiring permanent access to the master.

Replication in MySQL features support for one-way, asynchronous replication, in which one server acts as the master, while one or more other servers act as slaves. This is in contrast to the *synchronous* replication which is a characteristic of MySQL Cluster (see [MySQL Cluster](#)).

There are a number of solutions available for setting up replication between two servers, but the best method to use depends on the presence of data and the engine types you are using. For more information on the available options, see [How to Set Up Replication](#).

Replication is controlled through a number of different options and variables. These control the core operation of the replication, timeouts, and the databases and filters that can be applied on databases and tables. For more information on the available options, see [Replication and Binary Logging Options and Variables](#).

You can use replication to solve a number of different problems, including problems with performance, supporting the backup of different databases, and as part of a larger solution to alleviate system failures. For information on how to address these issues, see [Chapter 3, Replication Solutions](#).

For notes and tips on how different data types and statements are treated during replication, including details of replication features, version compatibility, upgrades, and problems and their resolution, including an FAQ, see [Chapter 4, Replication Notes and Tips](#).

For detailed information on the implementation of replication, how replication works, the process and contents of the binary log, background threads and the rules used to decide how statements are recorded and replication, see [Chapter 5, Replication Implementation](#).





---

## Chapter 2. Replication Configuration

### Table of Contents

How to Set Up Replication .....	4
Setting the Replication Master Configuration .....	5
Setting the Replication Slave Configuration .....	5
Creating a User for Replication .....	6
Obtaining the Replication Master Binary Log Coordinates .....	6
Creating a Data Snapshot Using <code>mysqldump</code> .....	7
Creating a Data Snapshot Using Raw Data Files .....	8
Setting Up Replication with New Master and Slaves .....	9
Setting Up Replication with Existing Data .....	10
Introducing Additional Slaves to an Existing Replication Environment .....	12
Setting the Master Configuration on the Slave .....	13
Replication and Binary Logging Options and Variables .....	13
Replication and Binary Logging Option and Variable Reference .....	14
Replication Master Options and Variables .....	17
Replication Slave Options and Variables .....	20
Binary Log Options and Variables .....	39
Common Replication Administration Tasks .....	44
Checking Replication Status .....	45
Pausing Replication on the Slave .....	46

Replication between servers in MySQL is based on the binary logging mechanism. The MySQL instance operating as the master (the source of the database changes) writes updates and changes as “events” to the binary log. The information in the binary log is stored in different logging formats according to the database changes being recorded. Slaves are configured to read the binary log from the master and to execute the events in the binary log on the slave’s local database.

The master is “dumb” in this scenario. Once binary logging has been enabled, all statements are recorded in the binary log. Each slave receives a copy of the entire contents of the binary log. It is the responsibility of the slave to decide which statements in the binary log should be executed; you cannot configure the master to log only certain events. If you do not specify otherwise, all events in the master binary log are executed on the slave. If required, you can configure the slave to process only events that apply to particular databases or tables.

Each slave keeps a record of the binary log coordinates: The file name and position within the file that it has read and processed from the master. This means that multiple slaves can be connected to the master and executing different parts of the same binary log. Because the slaves control this process, individual slaves can be connected and disconnected from the server without affecting the master’s operation. Also, because each slave remembers the position within the binary log, it is possible for slaves to be disconnected, reconnect and then “catch up” by continuing from the recorded position.

Both the master and each slave must be configured with a unique ID (using the `server-id` [13] option). In addition, each slave must be configured with information about the master host name, log file name, and position within that file. These details can be controlled from within a MySQL session using the `CHANGE MASTER TO` statement on the slave. The details are stored within the slave’s `master.info` file.

This section describes the setup and configuration required for a replication environment, including step-by-step instructions for creating a new replication environment. The major components of this section are:

- For a guide to setting up two or more servers for replication, [How to Set Up Replication](#), deals with the configuration of the systems and provides methods for copying data between the master and slaves.

- Detailed information on the different configuration options and variables that apply to replication is provided in [Replication and Binary Logging Options and Variables](#).
- Once started, the replication process should require little administration or monitoring. However, for advice on common tasks that you may want to execute, see [Common Replication Administration Tasks](#).

## How to Set Up Replication

This section describes how to set up complete replication of a MySQL server. There are a number of different methods for setting up replication, and the exact method to use depends on how you are setting up replication, and whether you already have data within your master database.

There are some generic tasks that are common to all replication setups:

- On the master, you must enable binary logging and configure a unique server ID. This might require a server restart. See [Setting the Replication Master Configuration](#).
- On each slave that you want to connect to the master, you must configure a unique server ID. This might require a server restart. See [Setting the Replication Slave Configuration](#).
- You may want to create a separate user that will be used by your slaves to authenticate with the master to read the binary log for replication. The step is optional. See [Creating a User for Replication](#).
- Before creating a data snapshot or starting the replication process, you should record the position of the binary log on the master. You will need this information when configuring the slave so that the slave knows where within the binary log to start executing events. See [Obtaining the Replication Master Binary Log Coordinates](#).
- If you already have data on your master and you want to use it to synchronize your slave, you will need to create a data snapshot. You can create a snapshot using `mysqldump` (see [Creating a Data Snapshot Using `mysqldump`](#)) or by copying the data files directly (see [Creating a Data Snapshot Using Raw Data Files](#)).
- You will need to configure the slave with settings for connecting to the master, such as the host name, login credentials, and binary log file name and position. See [Setting the Master Configuration on the Slave](#).

Once you have configured the basic options, you will need to follow the instructions for your replication setup. A number of alternatives are provided:

- If you are establishing a new MySQL master and one or more slaves, you need only set up the configuration, as you have no data to exchange. For guidance on setting up replication in this situation, see [Setting Up Replication with New Master and Slaves](#).
- If you are already running a MySQL server, and therefore already have data that must be transferred to your slaves before replication starts, have not previously configured the binary log and are able to shut down your MySQL server for a short period during the process, see [Setting Up Replication with Existing Data](#).
- If you are adding slaves to an existing replication environment, you can set up the slaves without affecting the master. See [Introducing Additional Slaves to an Existing Replication Environment](#).

If you will be administering MySQL replication servers, we suggest that you read this entire chapter through and try all statements mentioned in [SQL Statements for Controlling Master Servers](#), and [SQL Statements for Controlling Slave Servers](#). You should also familiarize yourself with the replication startup options described in [Replication and Binary Logging Options and Variables](#).

**Note**

Note that certain steps within the setup process require the [SUPER](#) privilege. If you do not have this privilege, it might not be possible to enable replication.

## Setting the Replication Master Configuration

On a replication master, you must enable binary logging and establish a unique server ID. If this has not already been done, this part of master setup requires a server restart.

Binary logging *must* be enabled on the master because the binary log is the basis for sending data changes from the master to its slaves. If binary logging is not enabled, replication will not be possible.

Each server within a replication group must be configured with a unique server ID. This ID is used to identify individual servers within the group, and must be a positive integer between 1 and  $(2^{32})-1$ . How you organize and select the numbers is entirely up to you.

To configure the binary log and server ID options, you will need to shut down your MySQL server and edit the `my.cnf` or `my.ini` file. Add the following options to the configuration file within the `[mysqld]` section. If these options already exist, but are commented out, uncomment the options and alter them according to your needs. For example, to enable binary logging using a log file name prefix of `mysql-bin`, and configure a server ID of 1, use these lines:

```
[mysqld]
log-bin=mysql-bin
server-id=1
```

After making the changes, restart the server.

**Note**

If you omit `server-id` [13] (or set it explicitly to its default value of 0), a master refuses connections from all slaves.

**Note**

For the greatest possible durability and consistency in a replication setup using [InnoDB](#) with transactions, you should use `innodb_flush_log_at_trx_commit=1` and `sync_binlog=1` in the master `my.cnf` file.

**Note**

Ensure that the `skip-networking` option is not enabled on your replication master. If networking has been disabled, your slave will not be able to communicate with the master and replication will fail.

## Setting the Replication Slave Configuration

On a replication slave, you must establish a unique server ID. If this has not already been done, this part of slave setup requires a server restart.

If the slave server ID is not already set, or the current value conflicts with the value that you have chosen for the master server, you should shut down your slave server and edit the configuration to specify a unique server ID. For example:

```
[mysqld]
```

```
server-id=2
```

After making the changes, restart the server.

If you are setting up multiple slaves, each one must have a unique `server-id` [13] value that differs from that of the master and from each of the other slaves. Think of `server-id` [13] values as something similar to IP addresses: These IDs uniquely identify each server instance in the community of replication partners.

**Note**

If you omit `server-id` [13] (or set it explicitly to its default value of 0), a slave refuses to connect to a master.

You do not have to enable binary logging on the slave for replication to be enabled. However, if you enable binary logging on the slave, you can use the binary log for data backups and crash recovery on the slave, and also use the slave as part of a more complex replication topology (for example, where the slave acts as a master to other slaves).

## Creating a User for Replication

Each slave must connect to the master using a MySQL user name and password, so there must be a user account on the master that the slave can use to connect. Any account can be used for this operation, providing it has been granted the `REPLICATION SLAVE` privilege. You may wish to create a different account for each slave, or connect to the master using the same account for each slave.

You need not create an account specifically for replication. However, you should be aware that the user name and password will be stored in plain text within the `master.info` file (see [Slave Status Logs](#)). Therefore, you may want to create a separate account that has privileges only for the replication process, to minimize the possibility of compromise to other accounts.

To create a new account, use `CREATE USER`. To grant this account the privileges required for replication, use the `GRANT` statement. If you create an account solely for the purposes of replication, that account needs only the `REPLICATION SLAVE` privilege. For example, to set up a new user, `repl`, that can connect for replication from any host within the `mydomain.com` domain, issue these statements on the master:

```
mysql> CREATE USER 'repl'@'%.mydomain.com' IDENTIFIED BY 'slavepass';
mysql> GRANT REPLICATION SLAVE ON *.* TO 'repl'@'%.mydomain.com';
```

See [Account Management Statements](#), for more information on statements for manipulation of user accounts.

## Obtaining the Replication Master Binary Log Coordinates

To configure replication on the slave you must determine the master's current coordinates within its binary log. You will need this information so that when the slave starts the replication process, it is able to start processing events from the binary log at the correct point.

If you have existing data on your master that you want to synchronize on your slaves before starting the replication process, you must stop processing statements on the master, and then obtain its current binary log coordinates and dump its data, before permitting the master to continue executing statements. If you do not stop the execution of statements, the data dump and the master status information that you use will not match and you will end up with inconsistent or corrupted databases on the slaves.

To obtain the master binary log coordinates, follow these steps:

1. Start a session on the master by connecting to it with the command-line client, and flush all tables and block write statements by executing the `FLUSH TABLES WITH READ LOCK` statement:

```
mysql> FLUSH TABLES WITH READ LOCK;
```

For `InnoDB` tables, note that `FLUSH TABLES WITH READ LOCK` also blocks `COMMIT` operations.

### Warning

Leave the client from which you issued the `FLUSH TABLES` statement running so that the read lock remains in effect. If you exit the client, the lock is released.

2. In a different session on the master, use the `SHOW MASTER STATUS` statement to determine the current binary log file name and position:

```
mysql > SHOW MASTER STATUS;
+-----+-----+-----+-----+
| File           | Position | Binlog_Do_DB | Binlog_Ignore_DB |
+-----+-----+-----+-----+
| mysql-bin.000003 | 73      | test        | manual,mysql      |
+-----+-----+-----+-----+
```

The `File` column shows the name of the log file and `Position` shows the position within the file. In this example, the binary log file is `mysql-bin.000003` and the position is 73. Record these values. You need them later when you are setting up the slave. They represent the replication coordinates at which the slave should begin processing new updates from the master.

If the master has been running previously without binary logging enabled, the log file name and position values displayed by `SHOW MASTER STATUS` or `mysqldump --master-data` will be empty. In that case, the values that you need to use later when specifying the slave's log file and position are the empty string ( `' '` ) and 4.

You now have the information you need to enable the slave to start reading from the binary log in the correct place to start replication.

If you have existing data that needs to be synchronized with the slave before you start replication, leave the client running so that the lock remains in place and then proceed to [Creating a Data Snapshot Using `mysqldump`](#), or [Creating a Data Snapshot Using Raw Data Files](#). The idea here is to prevent any further changes so that the data copied to the slaves is in synchrony with the master.

If you are setting up a brand new master and slave replication group, you can exit the first session to release the read lock.

## Creating a Data Snapshot Using `mysqldump`

One way to create a snapshot of the data in an existing master database is to use the `mysqldump` tool. Once the data dump has been completed, you then import this data into the slave before starting the replication process.

To obtain a snapshot of the data using `mysqldump`:

1. If you have not already locked the tables on the server to prevent statements that update data from executing:

Start a session on the server by connecting to it with the command-line client, and flush all tables and block write statements by executing the `FLUSH TABLES WITH READ LOCK` statement:

```
mysql> FLUSH TABLES WITH READ LOCK;
```

Remember to use `SHOW MASTER STATUS` and record the binary log details for use when starting up the slave. The point in time of your snapshot and the binary log position must match. See [Obtaining the Replication Master Binary Log Coordinates](#).

2. In another session, use `mysqldump` to create a dump either of all the databases you want to replicate, or of selected individual databases. For example:

```
shell> mysqldump --all-databases --lock-all-tables >dbdump.db
```

An alternative to using a bare dump, is to use the `--master-data` option, which automatically appends the `CHANGE MASTER TO` statement required on the slave to start the replication process.

```
shell> mysqldump --all-databases --master-data >dbdump.db
```

3. In the client where you acquired the read lock, release the lock:

```
mysql> UNLOCK TABLES;
```

Alternatively, exit the first session to release the read lock.

When choosing databases to include in the dump, remember that you will need to filter out databases on each slave that you do not want to include in the replication process.

You will need either to copy the dump file to the slave, or to use the file from the master when connecting remotely to the slave to import the data.

## Creating a Data Snapshot Using Raw Data Files

If your database is particularly large, copying the raw data files may be more efficient than using `mysqldump` and importing the file on each slave.

However, using this method with tables in storage engines with complex caching or logging algorithms may not give you a perfect “in time” snapshot as cache information and logging updates may not have been applied, even if you have acquired a global read lock. How the storage engine responds to this depends on its crash recovery abilities.

In addition, this method does not work reliably if the master and slave have different values for `ft_stopword_file`, `ft_min_word_len`, or `ft_max_word_len` and you are copying tables having full-text indexes.

If you are using `InnoDB` tables, you can use the `InnoDB Hot Backup` tool to obtain a consistent snapshot. This tool records the log name and offset corresponding to the snapshot to be later used on the slave. `Hot Backup` is a nonfree (commercial) tool that is not included in the standard MySQL distribution. See the `InnoDB Hot Backup` home page at <http://www.innodb.com/wp/products/hot-backup/> for detailed information.

Otherwise, you can obtain a reliable binary snapshot of `InnoDB` tables only after shutting down the MySQL Server.

To create a raw data snapshot of `MyISAM` tables you can use standard copy tools such as `cp` or `copy`, a remote copy tool such as `scp` or `rsync`, an archiving tool such as `zip` or `tar`, or a file system snapshot tool such as `dump`, providing that your MySQL data files exist on a single file system. If you are replicating only certain databases then make sure you copy only those files that related to those tables. (For `InnoDB`, all tables in all databases are stored in the shared tablespace files, unless you have the `innodb_file_per_table` option enabled.)

You may want to specifically exclude the following files from your archive:

- Files relating to the `mysql` database.
- The `master.info` file.
- The master's binary log files.
- Any relay log files.

To get the most consistent results with a raw data snapshot you should shut down the master server during the process, as follows:

1. Acquire a read lock and get the master's status. See [Obtaining the Replication Master Binary Log Coordinates](#).
2. In a separate session, shut down the master server:

```
shell> mysqladmin shutdown
```

3. Make a copy of the MySQL data files. The following examples show common ways to do this. You need to choose only one of them:

```
shell> tar cf /tmp/db.tar ./data
shell> zip -r /tmp/db.zip ./data
shell> rsync --recursive ./data /tmp/dbdata
```

4. Restart the master server.

If you are not using `InnoDB` tables, you can get a snapshot of the system from a master without shutting down the server as described in the following steps:

1. Acquire a read lock and get the master's status. See [Obtaining the Replication Master Binary Log Coordinates](#).
2. Make a copy of the MySQL data files. The following examples show common ways to do this. You need to choose only one of them:

```
shell> tar cf /tmp/db.tar ./data
shell> zip -r /tmp/db.zip ./data
shell> rsync --recursive ./data /tmp/dbdata
```

3. In the client where you acquired the read lock, release the lock:

```
mysql> UNLOCK TABLES;
```

Once you have created the archive or copy of the database, you will need to copy the files to each slave before starting the slave replication process.

## Setting Up Replication with New Master and Slaves

The easiest and most straightforward method for setting up replication is to use new master and slave servers.

You can also use this method if you are setting up new servers but have an existing dump of the databases from a different server that you want to load into your replication configuration. By loading the data into a new master, the data will be automatically replicated to the slaves.

To set up replication between a new master and slave:



1. Configure the MySQL master with the necessary configuration properties. See [Setting the Replication Master Configuration](#).
2. Start up the MySQL master.
3. Set up a user. See [Creating a User for Replication](#).
4. Obtain the master status information. See [Obtaining the Replication Master Binary Log Coordinates](#).
5. On the master, release the read lock:

```
mysql> UNLOCK TABLES;
```

6. On the slave, edit the MySQL configuration. See [Setting the Replication Slave Configuration](#).
7. Start up the MySQL slave.
8. Execute a `CHANGE MASTER TO` statement to set the master replication server configuration. See [Setting the Master Configuration on the Slave](#).

Perform the slave setup steps on each slave.

Because there is no data to load or exchange on a new server configuration you do not need to copy or import any information.

If you are setting up a new replication environment using the data from a different existing database server, you will now need to run the dump file generated from that server on the new master. The database updates will automatically be propagated to the slaves:

```
shell> mysql -h master < fulldb.dump
```

## Setting Up Replication with Existing Data

When setting up replication with existing data, you will need to decide how best to get the data from the master to the slave before starting the replication service.

The basic process for setting up replication with existing data is as follows:

1. With the MySQL master running, create a user to be used by the slave when connecting to the master during replication. See [Creating a User for Replication](#).
2. If you have not already configured the `server-id` [13] and enabled binary logging on the master server, you will need to shut it down to configure these options. See [Setting the Replication Master Configuration](#).

If you have to shut down your master server, this is a good opportunity to take a snapshot of its databases. You should obtain the master status (see [Obtaining the Replication Master Binary Log Coordinates](#)) before taking down the master, updating the configuration and taking a snapshot. For information on how to create a snapshot using raw data files, see [Creating a Data Snapshot Using Raw Data Files](#).

3. If your master server is already correctly configured, obtain its status (see [Obtaining the Replication Master Binary Log Coordinates](#)) and then use `mysqldump` to take a snapshot (see [Creating a Data Snapshot Using mysqldump](#)) or take a raw snapshot of the live server using the guide in [Creating a Data Snapshot Using Raw Data Files](#).
4. Update the configuration of the slave. See [Setting the Replication Slave Configuration](#).



5. The next step depends on how you created the snapshot of data on the master.

If you used `mysqldump`:

- a. Start the slave, using the `--skip-slave-start` [33] option so that replication does not start.
- b. Import the dump file:

```
shell> mysql < fulldb.dump
```

If you created a snapshot using the raw data files:

- a. Extract the data files into your slave data directory. For example:

```
shell> tar xvf dbdump.tar
```

You may need to set permissions and ownership on the files so that the slave server can access and modify them.

- b. Start the slave, using the `--skip-slave-start` [33] option so that replication does not start.
6. Configure the slave with the replication coordinates from the master. This tells the slave the binary log file and position within the file where replication needs to start. Also, configure the slave with the login credentials and host name of the master. For more information on the `CHANGE MASTER TO` statement required, see [Setting the Master Configuration on the Slave](#).
7. Start the slave threads:

```
mysql> START SLAVE;
```

After you have performed this procedure, the slave should connect to the master and catch up on any updates that have occurred since the snapshot was taken.

If you have forgotten to set the `server-id` [13] option for the master, slaves cannot connect to it.

If you have forgotten to set the `server-id` [13] option for the slave, you get the following error in the slave's error log:

```
Warning: You should set server-id to a non-0 value if master_host
is set; we will force server id to 2, but this MySQL server will
not act as a slave.
```

You also find error messages in the slave's error log if it is not able to replicate for any other reason.

Once a slave is replicating, you can find in its data directory one file named `master.info` and another named `relay-log.info`. The slave uses these two files to keep track of how much of the master's binary log it has processed. Do *not* remove or edit these files unless you know exactly what you are doing and fully understand the implications. Even in that case, it is preferred that you use the `CHANGE MASTER TO` statement to change replication parameters. The slave will use the values specified in the statement to update the status files automatically.

#### Note

The content of `master.info` overrides some of the server options specified on the command line or in `my.cnf`. See [Replication and Binary Logging Options and Variables](#), for more details.

A single snapshot of the master suffices for multiple slaves. To set up additional slaves, use the same master snapshot and follow the slave portion of the procedure just described.

## Introducing Additional Slaves to an Existing Replication Environment

To add another slave to an existing replication configuration, you can do so without stopping the master. Instead, set up the new slave by making a copy of an existing slave, except that you configure the new slave with a different `server-id` [13] value.

To duplicate an existing slave:

1. Shut down the existing slave:

```
shell> mysqladmin shutdown
```

2. Copy the data directory from the existing slave to the new slave. You can do this by creating an archive using `tar` or `WinZip`, or by performing a direct copy using a tool such as `cp` or `rsync`. Ensure that you also copy the log files and relay log files.

A common problem that is encountered when adding new replication slaves is that the new slave fails with a series of warning and error messages like these:

```
071118 16:44:10 [Warning] Neither --relay-log nor --relay-log-index were used; so
replication may break when this MySQL server acts as a slave and has his hostname
changed!! Please use '--relay-log=new_slave_hostname-relay-bin' to avoid this problem.
071118 16:44:10 [ERROR] Failed to open the relay log './old_slave_hostname-relay-bin.003525'
(relay_log_pos 22940879)
071118 16:44:10 [ERROR] Could not find target log during relay log initialization
071118 16:44:10 [ERROR] Failed to initialize the master info structure
```

This is due to the fact that, if the `--relay-log` [26] option is not specified, the relay log files contain the host name as part of their file names. (This is also true of the relay log index file if the `--relay-log-index` [26] option is not used. See [Replication and Binary Logging Options and Variables](#), for more information about these options.)

To avoid this problem, use the same value for `--relay-log` [26] on the new slave that was used on the existing slave. (If this option was not set explicitly on the existing slave, use `existing_slave_hostname-relay-bin`.) If this is not feasible, copy the existing slave's relay log index file to the new slave and set the `--relay-log-index` [26] option on the new slave to match what was used on the existing slave. (If this option was not set explicitly on the existing slave, use `existing_slave_hostname-relay-bin.index`.) Alternatively—if you have already tried to start the new slave (after following the remaining steps in this section) and have encountered errors like those described previously—then perform the following steps:

- a. If you have not already done so, issue a `STOP SLAVE` on the new slave.  
  
If you have already started the existing slave again, issue a `STOP SLAVE` on the existing slave as well.
  - b. Copy the contents of the existing slave's relay log index file into the new slave's relay log index file, making sure to overwrite any content already in the file.
  - c. Proceed with the remaining steps in this section.
3. Copy the `master.info` and `relay-log.info` files from the existing slave to the new slave if they were not located in the data directory. These files hold the current log coordinates for the master's binary log and the slave's relay log.
  4. Start the existing slave.
  5. On the new slave, edit the configuration and give the new slave a unique `server-id` [13] not used by the master or any of the existing slaves.

6. Start the new slave. The slave will use the information in its `master.info` file to start the replication process.

## Setting the Master Configuration on the Slave

To set up the slave to communicate with the master for replication, you must tell the slave the necessary connection information. To do this, execute the following statement on the slave, replacing the option values with the actual values relevant to your system:

```
mysql> CHANGE MASTER TO
->     MASTER_HOST='master_host_name',
->     MASTER_USER='replication_user_name',
->     MASTER_PASSWORD='replication_password',
->     MASTER_LOG_FILE='recorded_log_file_name',
->     MASTER_LOG_POS=recorded_log_position;
```

### Note

Replication cannot use Unix socket files. You must be able to connect to the master MySQL server using TCP/IP.

The `CHANGE MASTER TO` statement has other options as well. For example, it is possible to set up secure replication using SSL. For a full list of options, and information about the maximum permissible length for the string-valued options, see [CHANGE MASTER TO Syntax](#).

## Replication and Binary Logging Options and Variables

The next few sections contain information about `mysqld` options and server variables that are used in replication and for controlling the binary log. Options and variables for use on replication masters and replication slaves are covered separately, as are options and variables relating to binary logging. A set of quick-reference tables providing basic information about these options and variables is also included (in the next section following this one).

Of particular importance is the `--server-id [13]` option.

<b>Command-Line Format</b>	<code>--server-id=#</code>	
<b>Option-File Format</b>	<code>server-id</code>	
<b>Option Sets Variable</b>	Yes, <code>server_id</code>	
<b>Variable Name</b>	<code>server_id</code>	
<b>Variable Scope</b>	Global	
<b>Dynamic Variable</b>	Yes	
	<b>Permitted Values</b>	
	<b>Type</b>	<code>numeric</code>
	<b>Default</b>	<code>0</code>
	<b>Range</b>	<code>0 .. 4294967295</code>

This option is common to both master and slave replication servers, and is used in replication to enable master and slave servers to identify themselves uniquely. For additional information, see [Replication Master Options and Variables](#), and [Replication Slave Options and Variables](#).

On the master and each slave, you *must* use the `--server-id [13]` option to establish a unique replication ID in the range from 1 to  $2^{32} - 1$ . “Unique”, means that each ID must be different from every other ID in use by any other replication master or slave. Example: `server-id=3`.

If you omit `--server-id` [13], the default ID is 0, in which case a master refuses connections from all slaves, and a slave refuses to connect to a master. For more information, see [Setting the Replication Slave Configuration](#).

## Replication and Binary Logging Option and Variable Reference

The following tables list basic information about the MySQL command-line options and system variables applicable to replication and the binary log.

**Table 2.1. Replication Option/Variable Summary**

Name	Cmd-Line	Option file	System Var	Status Var	Var Scope	Dynamic
<a href="#">abort-slave-event-count</a>	Yes	Yes				
<a href="#">Com_change_master</a>				Yes	Both	No
<a href="#">Com_show_master_status</a>				Yes	Both	No
<a href="#">Com_show_new_master</a>				Yes	Both	No
<a href="#">Com_show_slave_hosts</a>				Yes	Both	No
<a href="#">Com_show_slave_status</a>				Yes	Both	No
<a href="#">Com_slave_start</a>				Yes	Both	No
<a href="#">Com_slave_stop</a>				Yes	Both	No
<a href="#">disconnect-slave-event-count</a>	Yes	Yes				
<a href="#">init_slave</a>	Yes	Yes	Yes		Global	Yes
<a href="#">log-slave-updates</a>	Yes	Yes			Global	No
- Variable: <a href="#">log_slave_updates</a>			Yes		Global	No
<a href="#">master-connect-retry</a>	Yes	Yes				
<a href="#">master-host</a>	Yes	Yes				
<a href="#">master-info-file</a>	Yes	Yes				
<a href="#">master-password</a>	Yes	Yes				
<a href="#">master-port</a>	Yes	Yes				
<a href="#">master-retry-count</a>	Yes	Yes				
<a href="#">master-ssl</a>	Yes	Yes				
<a href="#">master-ssl-ca</a>	Yes	Yes				
<a href="#">master-ssl-capath</a>	Yes	Yes				
<a href="#">master-ssl-cert</a>	Yes	Yes				
<a href="#">master-ssl-cipher</a>	Yes	Yes				
<a href="#">master-ssl-key</a>	Yes	Yes				
<a href="#">master-user</a>	Yes	Yes				
<a href="#">relay-log</a>	Yes	Yes			Global	No
- Variable: <a href="#">relay_log</a>			Yes		Global	No

Name	Cmd-Line	Option file	System Var	Status Var	Var Scope	Dynamic
<a href="#">relay-log-index</a>	Yes	Yes			Global	No
- Variable: <a href="#">relay_log_index</a>			Yes		Global	No
<a href="#">relay_log_index</a>	Yes	Yes	Yes		Global	No
<a href="#">relay-log-info-file</a>	Yes	Yes				
- Variable: <a href="#">relay_log_info_file</a>						
<a href="#">relay_log_info_file</a>	Yes	Yes	Yes		Global	No
<a href="#">relay_log_purge</a>	Yes	Yes	Yes		Global	Yes
<a href="#">relay_log_space_limit</a>	Yes	Yes	Yes		Global	No
<a href="#">replicate-do-db</a>	Yes	Yes				
<a href="#">replicate-do-table</a>	Yes	Yes				
<a href="#">replicate-ignore-db</a>	Yes	Yes				
<a href="#">replicate-ignore-table</a>	Yes	Yes				
<a href="#">replicate-rewrite-db</a>	Yes	Yes				
<a href="#">replicate-same-server-id</a>	Yes	Yes				
<a href="#">replicate-wild-do-table</a>	Yes	Yes				
<a href="#">replicate-wild-ignore-table</a>	Yes	Yes				
<a href="#">report-host</a>	Yes	Yes			Global	No
- Variable: <a href="#">report_host</a>			Yes		Global	No
<a href="#">report-password</a>	Yes	Yes			Global	No
- Variable: <a href="#">report_password</a>			Yes		Global	No
<a href="#">report-port</a>	Yes	Yes			Global	No
- Variable: <a href="#">report_port</a>			Yes		Global	No
<a href="#">report-user</a>	Yes	Yes			Global	No
- Variable: <a href="#">report_user</a>			Yes		Global	No
<a href="#">rpl_recovery_rank</a>			Yes		Global	Yes
<a href="#">Rpl_status</a>				Yes	Global	No
<a href="#">show-slave-auth-info</a>	Yes	Yes				
<a href="#">skip-slave-start</a>	Yes	Yes				

Name	Cmd-Line	Option file	System Var	Status Var	Var Scope	Dynamic
<a href="#">slave_compressed_protocol</a>	Yes	Yes	Yes		Global	Yes
<a href="#">slave-load-tmpdir</a>	Yes	Yes			Global	No
- Variable: <a href="#">slave_load_tmpdir</a>			Yes		Global	No
<a href="#">slave-net-timeout</a>	Yes	Yes			Global	Yes
- Variable: <a href="#">slave_net_timeout</a>			Yes		Global	Yes
<a href="#">Slave_open_temp_tables</a>				Yes	Global	No
<a href="#">Slave_retried_transactions</a>				Yes	Global	No
<a href="#">Slave_running</a>				Yes	Global	No
<a href="#">slave-skip-errors</a>	Yes	Yes			Global	No
- Variable: <a href="#">slave_skip_errors</a>			Yes		Global	No
<a href="#">slave_transaction_retries</a>	Yes	Yes	Yes		Global	Yes
<a href="#">sql_slave_skip_counter</a>			Yes		Global	Yes

[Replication Master Options and Variables](#), provides more detailed information about options and variables relating to replication master servers. For more information about options and variables relating to replication slaves, see [Replication Slave Options and Variables](#).

**Table 2.2. Binary Logging Option/Variable Summary**

Name	Cmd-Line	Option file	System Var	Status Var	Var Scope	Dynamic
<a href="#">Binlog_cache_disk_use</a>				Yes	Global	No
<a href="#">binlog_cache_size</a>	Yes	Yes	Yes		Global	Yes
<a href="#">Binlog_cache_use</a>				Yes	Global	No
<a href="#">binlog-do-db</a>	Yes	Yes				
<a href="#">binlog-ignore-db</a>	Yes	Yes				
<a href="#">Com_show_binlog_events</a>				Yes	Both	No
<a href="#">Com_show_binlogs</a>				Yes	Both	No
<a href="#">max_binlog_cache_size</a>	Yes	Yes	Yes		Global	Yes
<a href="#">max-binlog-dump-events</a>	Yes	Yes				
<a href="#">max_binlog_size</a>	Yes	Yes	Yes		Global	Yes
<a href="#">sporadic-binlog-dump-fail</a>	Yes	Yes				

[Binary Log Options and Variables](#), provides more detailed information about options and variables relating to binary logging. For additional general information about the binary log, see [The Binary Log](#).

For a table showing *all* command-line options, system and status variables used with `mysqld`, see [Server Option and Variable Reference](#).

## Replication Master Options and Variables

This section describes the server options and system variables that you can use on replication master servers. You can specify the options either on the [command line](#) or in an [option file](#). You can specify system variable values using [SET](#).

On the master and each slave, you must use the [server-id](#) [13] option to establish a unique replication ID. For each server, you should pick a unique positive integer in the range from 1 to  $2^{32} - 1$ , and each ID must be different from every other ID in use by any other replication master or slave. Example: [server-id=3](#).

For options used on the master for controlling binary logging, see [Binary Log Options and Variables](#).

- [auto\\_increment\\_increment](#) [17]

<b>Version Introduced</b>	5.0.2
<b>Command-Line Format</b>	<code>--auto-increment-increment[=#]</code>
<b>Option-File Format</b>	<code>auto_increment_increment</code>
<b>Option Sets Variable</b>	Yes, <a href="#">auto_increment_increment</a> [17]
<b>Variable Name</b>	<code>auto_increment_increment</code>
<b>Variable Scope</b>	Global, Session
<b>Dynamic Variable</b>	Yes
	<b>Permitted Values</b>
	<b>Type</b> <code>numeric</code>
	<b>Default</b> <code>1</code>
	<b>Range</b> <code>1 .. 65535</code>

[auto\\_increment\\_increment](#) [17] and [auto\\_increment\\_offset](#) [20] are intended for use with master-to-master replication, and can be used to control the operation of `AUTO_INCREMENT` columns. Both variables have global and session values, and each can assume an integer value between 1 and 65,535 inclusive. Setting the value of either of these two variables to 0 causes its value to be set to 1 instead. Attempting to set the value of either of these two variables to an integer greater than 65,535 or less than 0 causes its value to be set to 65,535 instead. Attempting to set the value of [auto\\_increment\\_increment](#) [17] or [auto\\_increment\\_offset](#) [20] to a noninteger value gives rise to an error, and the actual value of the variable remains unchanged.

These two variables affect `AUTO_INCREMENT` column behavior as follows:

- [auto\\_increment\\_increment](#) [17] controls the interval between successive column values. For example:

```
mysql> SHOW VARIABLES LIKE 'auto_inc%';
+-----+-----+
| Variable_name | Value |
+-----+-----+
| auto_increment_increment | 1 |
| auto_increment_offset | 1 |
+-----+-----+
2 rows in set (0.00 sec)

mysql> CREATE TABLE autoincl
-> (col INT NOT NULL AUTO_INCREMENT PRIMARY KEY);
Query OK, 0 rows affected (0.04 sec)
```

```
mysql> SET @@auto_increment_increment=10;
Query OK, 0 rows affected (0.00 sec)

mysql> SHOW VARIABLES LIKE 'auto_inc%';
+-----+-----+
| Variable_name | Value |
+-----+-----+
| auto_increment_increment | 10 |
| auto_increment_offset | 1 |
+-----+-----+
2 rows in set (0.01 sec)

mysql> INSERT INTO autoinc1 VALUES (NULL), (NULL), (NULL), (NULL);
Query OK, 4 rows affected (0.00 sec)
Records: 4 Duplicates: 0 Warnings: 0

mysql> SELECT col FROM autoinc1;
+-----+
| col |
+-----+
| 1 |
| 11 |
| 21 |
| 31 |
+-----+
4 rows in set (0.00 sec)
```

- `auto_increment_offset` [20] determines the starting point for the `AUTO_INCREMENT` column value. Consider the following, assuming that these statements are executed during the same session as the example given in the description for `auto_increment_increment` [17]:

```
mysql> SET @@auto_increment_offset=5;
Query OK, 0 rows affected (0.00 sec)

mysql> SHOW VARIABLES LIKE 'auto_inc%';
+-----+-----+
| Variable_name | Value |
+-----+-----+
| auto_increment_increment | 10 |
| auto_increment_offset | 5 |
+-----+-----+
2 rows in set (0.00 sec)

mysql> CREATE TABLE autoinc2
  -> (col INT NOT NULL AUTO_INCREMENT PRIMARY KEY);
Query OK, 0 rows affected (0.06 sec)

mysql> INSERT INTO autoinc2 VALUES (NULL), (NULL), (NULL), (NULL);
Query OK, 4 rows affected (0.00 sec)
Records: 4 Duplicates: 0 Warnings: 0

mysql> SELECT col FROM autoinc2;
+-----+
| col |
+-----+
| 5 |
| 15 |
| 25 |
| 35 |
+-----+
4 rows in set (0.02 sec)
```

If the value of `auto_increment_offset` [20] is greater than that of `auto_increment_increment` [17], the value of `auto_increment_offset` [20] is ignored.



Should one or both of these variables be changed and then new rows inserted into a table containing an `AUTO_INCREMENT` column, the results may seem counterintuitive because the series of `AUTO_INCREMENT` values is calculated without regard to any values already present in the column, and the next value inserted is the least value in the series that is greater than the maximum existing value in the `AUTO_INCREMENT` column. In other words, the series is calculated like so:

$$\text{auto\_increment\_offset} + N \times \text{auto\_increment\_increment}$$

where  $N$  is a positive integer value in the series [1, 2, 3, ...]. For example:

```
mysql> SHOW VARIABLES LIKE 'auto_inc%';
+-----+
| Variable_name | Value |
+-----+
| auto_increment_increment | 10 |
| auto_increment_offset | 5 |
+-----+
2 rows in set (0.00 sec)

mysql> SELECT col FROM autoinc1;
+-----+
| col |
+-----+
| 1 |
| 11 |
| 21 |
| 31 |
+-----+
4 rows in set (0.00 sec)

mysql> INSERT INTO autoinc1 VALUES (NULL), (NULL), (NULL), (NULL);
Query OK, 4 rows affected (0.00 sec)
Records: 4 Duplicates: 0 Warnings: 0

mysql> SELECT col FROM autoinc1;
+-----+
| col |
+-----+
| 1 |
| 11 |
| 21 |
| 31 |
| 35 |
| 45 |
| 55 |
| 65 |
+-----+
8 rows in set (0.00 sec)
```

The values shown for `auto_increment_increment` [17] and `auto_increment_offset` [20] generate the series  $5 + N \times 10$ , that is, [5, 15, 25, 35, 45, ...]. The greatest value present in the `col` column prior to the `INSERT` is 31, and the next available value in the `AUTO_INCREMENT` series is 35, so the inserted values for `col` begin at that point and the results are as shown for the `SELECT` query.

It is not possible to confine the effects of these two variables to a single table, and thus they do not take the place of the sequences offered by some other database management systems; these variables control the behavior of all `AUTO_INCREMENT` columns in *all* tables on the MySQL server. If the global value of either variable is set, its effects persist until the global value is changed or overridden by setting the session value, or until `mysqld` is restarted. If the local value is set, the new value affects `AUTO_INCREMENT` columns for all tables into which new rows are inserted by the current user for the duration of the session, unless the values are changed during that session.

The `auto_increment_increment` [17] variable was added in MySQL 5.0.2. Its default value is 1. See [Replication and AUTO\\_INCREMENT](#).

`auto_increment_increment` [17] is supported for use with `NDB` tables beginning with MySQL 5.0.46. Previously, setting it when using MySQL Cluster tables produced unpredictable results.

- `auto_increment_offset` [20]

<b>Version Introduced</b>	5.0.2	
<b>Command-Line Format</b>	<code>--auto-increment-offset [=#]</code>	
<b>Option-File Format</b>	<code>auto_increment_offset</code>	
<b>Option Sets Variable</b>	Yes, <code>auto_increment_offset</code> [20]	
<b>Variable Name</b>	<code>auto_increment_offset</code>	
<b>Variable Scope</b>	Global, Session	
<b>Dynamic Variable</b>	Yes	
	<b>Permitted Values</b>	
	<b>Type</b>	<code>numeric</code>
	<b>Default</b>	<code>1</code>
	<b>Range</b>	<code>1 .. 65535</code>

This variable was introduced in MySQL 5.0.2. Its default value is 1. For particulars, see the description for `auto_increment_increment` [17].

`auto_increment_offset` [20] is supported for use with `NDB` tables beginning with MySQL 5.0.46. Previously, setting it when using MySQL Cluster tables produced unpredictable results.

## Replication Slave Options and Variables

This section describes the server options and system variables that apply to slave replication servers. You can specify the options either on the [command line](#) or in an [option file](#). Many of the options can be set while the server is running by using the `CHANGE MASTER TO` statement. You can specify system variable values using `SET`.

**Server ID.** On the master and each slave, you must use the `server-id` [13] option to establish a unique replication ID in the range from 1 to  $2^{32} - 1$ . “Unique” means that each ID must be different from every other ID in use by any other replication master or slave. Example `my.cnf` file:

```
[mysqld]
server-id=3
```

Some slave server replication options are handled in a special way, in the sense that each is ignored if a `master.info` file exists when the slave starts and contains a value for the option. The following options are handled this way:

- `--master-host` [24]
- `--master-user` [25]
- `--master-password` [24]
- `--master-port` [24]

- `--master-connect-retry` [23]
- `--master-ssl` [25]
- `--master-ssl-ca` [25]
- `--master-ssl-capath` [25]
- `--master-ssl-cert` [25]
- `--master-ssl-cipher` [25]
- `--master-ssl-key` [25]

The `master.info` file format in MySQL 5.0 includes as its first line the number of lines in the file. (See [Replication Relay and Status Logs](#).) If you upgrade an older server (before MySQL 4.1.1) to a newer version, the new server upgrades the `master.info` file to the new format automatically when it starts. However, if you downgrade a newer server to a version older than 4.1.1, you should manually remove the first line before starting the older server for the first time. Note that, in this case, the downgraded server can no longer use an SSL connection to communicate with the master.

If no `master.info` file exists when the slave server starts, it uses the values for those options that are specified in option files or on the command line. This occurs when you start the server as a replication slave for the very first time, or when you have run `RESET SLAVE` and then have shut down and restarted the slave.

If the `master.info` file exists when the slave server starts, the server uses its contents and ignores any startup options that correspond to the values listed in the file. Thus, if you start the slave server with different values of the startup options that correspond to values in the `master.info` file, the different values have no effect because the server continues to use the `master.info` file. To use different values, the preferred method is to use the `CHANGE MASTER TO` statement to reset the values while the slave is running. Alternatively, you can stop the server, remove the `master.info` file, and restart the server with different option values.

Suppose that you specify this option in your `my.cnf` file:

```
[mysqld]
master-host=some_host
```

The first time you start the server as a replication slave, it reads and uses that option from the `my.cnf` file. The server then records the value in the `master.info` file. The next time you start the server, it reads the master host value from the `master.info` file only and ignores the value in the option file. If you modify the `my.cnf` file to specify a different master host of `some_other_host`, the change still has no effect. You should use `CHANGE MASTER TO` instead.

Because the server gives an existing `master.info` file precedence over the startup options just described, you might prefer not to use startup options for these values at all, and instead specify them by using the `CHANGE MASTER TO` statement. See [CHANGE MASTER TO Syntax](#).

This example shows a more extensive use of startup options to configure a slave server:

```
[mysqld]
server-id=2
master-host=db-master.mycompany.com
master-port=3306
master-user=pertinax
master-password=freitag
master-connect-retry=60
```

```
report-host=db-slave.mycompany.com
```

**Startup options for replication slaves.** The following list describes startup options for controlling replication slave servers. Many of these options can be set while the server is running by using the `CHANGE MASTER TO` statement. Others, such as the `--replicate-*` options, can be set only when the slave server starts. Replication-related system variables are discussed later in this section.

- `--abort-slave-event-count` [22]

<b>Command-Line Format</b>	<code>--abort-slave-event-count=#</code>	
<b>Option-File Format</b>	<code>abort-slave-event-count</code>	
	<b>Permitted Values</b>	
	<b>Type</b>	numeric
	<b>Default</b>	0
	<b>Min Value</b>	0

When this option is set to some positive integer *value* other than 0 (the default) it affects replication behavior as follows: After the slave SQL thread has started, *value* log events are permitted to be executed; after that, the slave SQL thread does not receive any more events, just as if the network connection from the master were cut. The slave thread continues to run, and the output from `SHOW SLAVE STATUS` displays `Yes` in both the `Slave_IO_Running` and the `Slave_SQL_Running` columns, but no further events are read from the relay log.

This option is used internally by the MySQL test suite for replication testing and debugging. It is not intended for use in a production setting.

- `--disconnect-slave-event-count` [22]

<b>Command-Line Format</b>	<code>--disconnect-slave-event-count=#</code>	
<b>Option-File Format</b>	<code>disconnect-slave-event-count</code>	
	<b>Permitted Values</b>	
	<b>Type</b>	numeric
	<b>Default</b>	0

This option is used internally by the MySQL test suite for replication testing and debugging.

- `--log-slave-updates` [22]

<b>Command-Line Format</b>	<code>--log-slave-updates</code>	
<b>Option-File Format</b>	<code>log-slave-updates</code>	
<b>Option Sets Variable</b>	Yes, <code>log_slave_updates</code> [43]	
<b>Variable Name</b>	<code>log_slave_updates</code>	
<b>Variable Scope</b>	Global	
<b>Dynamic Variable</b>	No	
	<b>Permitted Values</b>	
	<b>Type</b>	boolean
	<b>Default</b>	<code>FALSE</code>

Normally, a slave does not log to its own binary log any updates that are received from a master server. This option tells the slave to log the updates performed by its SQL thread to its own binary log. For this option to have any effect, the slave must also be started with the `--log-bin` [39] option to enable binary logging. `--log-slave-updates` [22] is used when you want to chain replication servers. For example, you might want to set up replication servers using this arrangement:

```
A --> B --> C
```

Here, **A** serves as the master for the slave **B**, and **B** serves as the master for the slave **C**. For this to work, **B** must be both a master *and* a slave. You must start both **A** and **B** with `--log-bin` [39] to enable binary logging, and **B** with the `--log-slave-updates` [22] option so that updates received from **A** are logged by **B** to its binary log.

- `--log-warnings[=level]`

<b>Command-Line Format</b>	<code>--log-warnings[=#]</code>	
	<code>-W [#]</code>	
<b>Option-File Format</b>	<code>log-warnings</code>	
<b>Option Sets Variable</b>	Yes, <code>log_warnings</code>	
<b>Variable Name</b>	<code>log_warnings</code>	
<b>Variable Scope</b>	Global, Session	
<b>Dynamic Variable</b>	Yes	
<b>Disabled by</b>	<code>skip-log-warnings</code>	
	<b>Permitted Values</b>	
	<b>Platform Bit Size</b>	64
	<b>Type</b>	numeric
	<b>Default</b>	1
	<b>Range</b>	0 .. 18446744073709547520

This option causes a server to print more messages to the error log about what it is doing. With respect to replication, the server generates warnings that it succeeded in reconnecting after a network/connection failure, and informs you as to how each slave thread started. This option is enabled by default; to disable it, use `--skip-log-warnings`. Aborted connections are not logged to the error log unless the value is greater than 1.

Note that the effects of this option are not limited to replication. It produces warnings across a spectrum of server activities.

- `--master-connect-retry=seconds` [23]

<b>Command-Line Format</b>	<code>--master-connect-retry=#</code>	
<b>Option-File Format</b>	<code>master-connect-retry</code>	
<b>Deprecated</b>	5.1.17	
	<b>Permitted Values</b>	
	<b>Type</b>	numeric
	<b>Default</b>	60

The number of seconds that the slave thread sleeps before trying to reconnect to the master in case the master goes down or the connection is lost. The value in the `master.info` file takes precedence if it can be read. If not set, the default is 60. Connection retries are not invoked until the slave times out reading data from the master according to the value of `--slave-net-timeout` [34]. The number of reconnection attempts is limited by the `--master-retry-count` [25] option.

- `--master-host=host_name` [24]

<b>Command-Line Format</b>	<code>--master-host=name</code>	
<b>Option-File Format</b>	<code>master-host</code>	
<b>Deprecated</b>	5.1.17	
	<b>Permitted Values</b>	
	<b>Type</b>	<code>string</code>

The host name or IP address of the master replication server. The value in `master.info` takes precedence if it can be read. If no master host is specified, the slave thread does not start.

- `--master-info-file=file_name` [24]

<b>Command-Line Format</b>	<code>--master-info-file=file_name</code>	
<b>Option-File Format</b>	<code>master-info-file=file_name</code>	
	<b>Permitted Values</b>	
	<b>Type</b>	<code>file name</code>
	<b>Default</b>	<code>master.info</code>

The name to use for the file in which the slave records information about the master. The default name is `master.info` in the data directory. For information about the format of this file, see [Slave Status Logs](#).

- `--master-password=password` [24]

<b>Command-Line Format</b>	<code>--master-password=name</code>	
<b>Option-File Format</b>	<code>master-password</code>	
<b>Deprecated</b>	5.1.17	
	<b>Permitted Values</b>	
	<b>Type</b>	<code>string</code>

The password of the account that the slave thread uses for authentication when it connects to the master. The value in the `master.info` file takes precedence if it can be read. If not set, an empty password is assumed.

- `--master-port=port_number` [24]

<b>Command-Line Format</b>	<code>--master-port=#</code>	
<b>Option-File Format</b>	<code>master-port</code>	
<b>Deprecated</b>	5.1.17	
	<b>Permitted Values</b>	
	<b>Type</b>	<code>numeric</code>

	<b>Default</b>	3306
--	----------------	------

The TCP/IP port number that the master is listening on. The value in the `master.info` file takes precedence if it can be read. If not set, the compiled-in setting is assumed (normally 3306).

- `--master-retry-count=count` [25]

<b>Command-Line Format</b>	<code>--master-retry-count=#</code>	
<b>Option-File Format</b>	<code>master-retry-count</code>	
<b>Deprecated</b>	5.6.1	
	<b>Permitted Values</b>	
	<b>Platform Bit Size</b>	32
	<b>Type</b>	numeric
	<b>Default</b>	86400
	<b>Range</b>	0 .. 4294967295
	<b>Permitted Values</b>	
	<b>Platform Bit Size</b>	64
	<b>Type</b>	numeric
	<b>Default</b>	86400
	<b>Range</b>	0 .. 18446744073709551615

The number of times that the slave tries to connect to the master before giving up. Reconnects are attempted at intervals set by the `--master-connect-retry` [23] option (or the `MASTER_CONNECT_RETRY` option of the `CHANGE MASTER TO` statement) and reconnects are triggered when data reads by the slave time out according to the `--slave-net-timeout` [34] option. The default value is 86400. A value of 0 means “infinite”; the slave attempts to connect forever.

- `--master-ssl` [25], `--master-ssl-ca=file_name`, `--master-ssl-capath=directory_name`, `--master-ssl-cert=file_name`, `--master-ssl-cipher=cipher_list`, `--master-ssl-key=file_name`

These options are used for setting up a secure replication connection to the master server using SSL. Their meanings are the same as the corresponding `--ssl`, `--ssl-ca`, `--ssl-capath`, `--ssl-cert`, `--ssl-cipher`, `--ssl-key` options that are described in [SSL Command Options](#). The values in the `master.info` file take precedence if they can be read.

- `--master-user=user_name` [25]

<b>Command-Line Format</b>	<code>--master-user=name</code>	
<b>Option-File Format</b>	<code>master-user</code>	
<b>Deprecated</b>	5.1.17	
	<b>Permitted Values</b>	
	<b>Type</b>	string
	<b>Default</b>	test

The user name of the account that the slave thread uses for authentication when it connects to the master. This account must have the `REPLICATION SLAVE` privilege. The value in the `master.info` file takes precedence if it can be read. If the master user name is not set, the name `test` is assumed.

- `--max-relay-log-size=size` [26]

The size at which the server rotates relay log files automatically. For more information, see [Replication Relay and Status Logs](#). The default size is 1GB.

- `--read-only` [26]

Cause the slave to permit no updates except from slave threads or from users having the `SUPER` privilege. On a slave server, this can be useful to ensure that the slave accepts updates only from its master server and not from clients. This variable does not apply to `TEMPORARY` tables.

- `--relay-log=file_name` [26]

<b>Command-Line Format</b>	<code>--relay-log=name</code>	
<b>Option-File Format</b>	<code>relay-log</code>	
<b>Option Sets Variable</b>	Yes, <code>relay_log</code> [35]	
<b>Variable Name</b>	<code>relay-log</code>	
<b>Variable Scope</b>	Global	
<b>Dynamic Variable</b>	No	
	<b>Permitted Values</b>	
	<b>Type</b>	<code>file name</code>

The basename for the relay log. The default basename is `host_name-relay-bin`. The server writes the file in the data directory unless the basename is given with a leading absolute path name to specify a different directory. The server creates relay log files in sequence by adding a numeric suffix to the basename.

Due to the manner in which MySQL parses server options, if you specify this option, you must supply a value; *the default basename is used only if the option is not actually specified*. If you use the `--relay-log` [26] option without specifying a value, unexpected behavior is likely to result; this behavior depends on the other options used, the order in which they are specified, and whether they are specified on the command line or in an option file. For more information about how MySQL handles server options, see [Specifying Program Options](#).

If you specify this option, the value specified is also used as the basename for the relay log index file. You can override this behavior by specifying a different relay log index file basename using the `--relay-log-index` [26] option.

You may find the `--relay-log` [26] option useful in performing the following tasks:

- Creating relay logs whose names are independent of host names.
- If you need to put the relay logs in some area other than the data directory because your relay logs tend to be very large and you do not want to decrease `max_relay_log_size`.
- To increase speed by using load-balancing between disks.
- `--relay-log-index=file_name` [26]



<b>Command-Line Format</b>	<code>--relay-log-index=name</code>	
<b>Option-File Format</b>	<code>relay-log-index</code>	
<b>Option Sets Variable</b>	Yes, <code>relay_log_index</code> [36]	
<b>Variable Name</b>	<code>relay-log-index</code>	
<b>Variable Scope</b>	Global	
<b>Dynamic Variable</b>	No	
	<b>Permitted Values</b>	
	<b>Type</b>	<code>file name</code>

The name to use for the relay log index file. The default name is `host_name-relay-bin.index` in the data directory, where `host_name` is the name of the slave server.

Due to the manner in which MySQL parses server options, if you specify this option, you must supply a value; *the default basename is used only if the option is not actually specified*. If you use the `--relay-log-index` [26] option without specifying a value, unexpected behavior is likely to result; this behavior depends on the other options used, the order in which they are specified, and whether they are specified on the command line or in an option file. For more information about how MySQL handles server options, see [Specifying Program Options](#).

If you specify this option, the value specified is also used as the basename for the relay logs. You can override this behavior by specifying a different relay log file basename using the `--relay-log` [26] option.

- `--relay-log-info-file=file_name` [27]

<b>Command-Line Format</b>	<code>--relay-log-info-file=file_name</code>	
<b>Option-File Format</b>	<code>relay-log-info-file</code>	
<b>Option Sets Variable</b>	Yes, <code>relay_log_info_file</code> [36]	
	<b>Permitted Values</b>	
	<b>Type</b>	<code>file name</code>
	<b>Default</b>	<code>relay-log.info</code>

The name to use for the file in which the slave records information about the relay logs. The default name is `relay-log.info` in the data directory. For information about the format of this file, see [Slave Status Logs](#).

- `--relay-log-purge={0|1}` [27]

Disable or enable automatic purging of relay logs as soon as they are no longer needed. The default value is 1 (enabled). This is a global variable that can be changed dynamically with `SET GLOBAL relay_log_purge = N`.

- `--relay-log-space-limit=size` [27]

This option places an upper limit on the total size in bytes of all relay logs on the slave. A value of 0 means “no limit.” This is useful for a slave server host that has limited disk space. When the limit is reached, the I/O thread stops reading binary log events from the master server until the SQL thread has caught up and deleted some unused relay logs. Note that this limit is not absolute: There are cases where the SQL thread needs more events before it can delete relay logs. In that case, the I/O thread exceeds the limit until it becomes possible for the SQL thread to delete some relay logs because not

doing so would cause a deadlock. You should not set `--relay-log-space-limit` [27] to less than twice the value of `--max-relay-log-size` [26] (or `--max-binlog-size` [43] if `--max-relay-log-size` [26] is 0). In that case, there is a chance that the I/O thread waits for free space because `--relay-log-space-limit` [27] is exceeded, but the SQL thread has no relay log to purge and is unable to satisfy the I/O thread. This forces the I/O thread to ignore `--relay-log-space-limit` [27] temporarily.

- `--replicate-do-db=db_name` [28]

<b>Command-Line Format</b>	<code>--replicate-do-db=name</code>	
<b>Option-File Format</b>	<code>replicate-do-db</code>	
	<b>Permitted Values</b>	
	<b>Type</b>	<code>string</code>

Tell the slave SQL thread to restrict replication to statements where the default database (that is, the one selected by `USE`) is `db_name`. To specify more than one database, use this option multiple times, once for each database. Note that this does not replicate cross-database statements such as `UPDATE some_db.some_table SET foo='bar'` while having selected a different database or no database.

#### Warning

To specify multiple databases you *must* use multiple instances of this option. Because database names can contain commas, if you supply a comma separated list then the list will be treated as the name of a single database.

An example of what does not work as you might expect: If the slave is started with `--replicate-do-db=sales` [28] and you issue the following statements on the master, the `UPDATE` statement is *not* replicated:

```
USE prices;
UPDATE sales.january SET amount=amount+1000;
```

The main reason for this “check just the default database” behavior is that it is difficult from the statement alone to know whether it should be replicated (for example, if you are using multiple-table `DELETE` or multiple-table `UPDATE` statements that go across multiple databases). It is also faster to check only the default database rather than all databases if there is no need.

If you need cross-database updates to work, use `--replicate-wild-do-table=db_name.%` [30] instead. See [How Servers Evaluate Replication Filtering Rules](#).

Beginning with MySQL 5.0.84, this option has no effect on `BEGIN`, `COMMIT`, or `ROLLBACK` statements. (Bug #43263)

- `--replicate-ignore-db=db_name` [28]

<b>Command-Line Format</b>	<code>--replicate-ignore-db=name</code>	
<b>Option-File Format</b>	<code>replicate-ignore-db</code>	
	<b>Permitted Values</b>	
	<b>Type</b>	<code>string</code>

Tells the slave SQL thread not to replicate any statement where the default database (that is, the one selected by `USE`) is `db_name`. To specify more than one database to ignore, use this option multiple times, once for each database. You should *not* use this option if you are using cross-database updates

and you do not want these updates to be replicated. See [How Servers Evaluate Replication Filtering Rules](#).

An example of what does not work as you might expect: If the slave is started with `--replicate-ignore-db=sales` [28] and you issue the following statements on the master, the `UPDATE` statement *is* replicated:

```
USE prices;
UPDATE sales.january SET amount=amount+1000;
```

#### Note

In the preceding example the statement is replicated because `--replicate-ignore-db` [28] only applies to the default database (set through the `USE` statement). Because the `sales` database was specified explicitly in the statement, the statement has not been filtered.

If you need cross-database updates to work, use `--replicate-wild-ignore-table=db_name.%` [31] instead. See [How Servers Evaluate Replication Filtering Rules](#).

Beginning with MySQL 5.0.84, this option has no effect on `BEGIN`, `COMMIT`, or `ROLLBACK` statements. (Bug #43263)

- `--replicate-do-table=db_name.tbl_name` [29]

<b>Command-Line Format</b>	<code>--replicate-do-table=name</code>	
<b>Option-File Format</b>	<code>replicate-do-table</code>	
	<b>Permitted Values</b>	
	<b>Type</b>	<code>string</code>

Tells the slave SQL thread to restrict replication to the specified table. To specify more than one table, use this option multiple times, once for each table. This works for both cross-database updates and default database updates, in contrast to `--replicate-do-db` [28]. See [How Servers Evaluate Replication Filtering Rules](#).

This option affects only statements that apply to tables. It does not affect statements that apply only to other database objects, such as stored routines. To filter statements operating on stored routines, use one or more of the `--replicate-*-db` options.

- `--replicate-ignore-table=db_name.tbl_name` [29]

<b>Command-Line Format</b>	<code>--replicate-ignore-table=name</code>	
<b>Option-File Format</b>	<code>replicate-ignore-table</code>	
	<b>Permitted Values</b>	
	<b>Type</b>	<code>string</code>

Tells the slave SQL thread not to replicate any statement that updates the specified table, even if any other tables might be updated by the same statement. To specify more than one table to ignore, use this option multiple times, once for each table. This works for cross-database updates, in contrast to `--replicate-ignore-db` [28]. See [How Servers Evaluate Replication Filtering Rules](#).

This option affects only statements that apply to tables. It does not affect statements that apply only to other database objects, such as stored routines. To filter statements operating on stored routines, use one or more of the `--replicate-*-db` options.

- `--replicate-rewrite-db=from_name->to_name` [30]

<b>Command-Line Format</b>	<code>--replicate-rewrite-db=old_name-&gt;new_name</code>	
<b>Option-File Format</b>	<code>replicate-rewrite-db</code>	
	<b>Permitted Values</b>	
	<b>Type</b>	string

Tells the slave to translate the default database (that is, the one selected by `USE`) to *to\_name* if it was *from\_name* on the master. Only statements involving tables are affected (not statements such as `CREATE DATABASE`, `DROP DATABASE`, and `ALTER DATABASE`), and only if *from\_name* is the default database on the master. This does not work for cross-database updates. To specify multiple rewrites, use this option multiple times. The server uses the first one with a *from\_name* value that matches. The database name translation is done *before* the `--replicate-*` rules are tested.

If you use this option on the command line and the “>” character is special to your command interpreter, quote the option value. For example:

```
shell> mysqld --replicate-rewrite-db="olddb->newdb"
```

- `--replicate-same-server-id` [30]

<b>Version Introduced</b>	5.0.1	
<b>Command-Line Format</b>	<code>--replicate-same-server-id</code>	
<b>Option-File Format</b>	<code>replicate-same-server-id</code>	
	<b>Permitted Values</b>	
	<b>Type</b>	boolean
	<b>Default</b>	FALSE

To be used on slave servers. Usually you should use the default setting of 0, to prevent infinite loops caused by circular replication. If set to 1, the slave does not skip events having its own server ID. Normally, this is useful only in rare configurations. Cannot be set to 1 if `--log-slave-updates` [22] is used. By default, the slave I/O thread does not write binary log events to the relay log if they have the slave's server ID (this optimization helps save disk usage). If you want to use `--replicate-same-server-id` [30], be sure to start the slave with this option before you make the slave read its own events that you want the slave SQL thread to execute.

- `--replicate-wild-do-table=db_name.tbl_name` [30]

<b>Command-Line Format</b>	<code>--replicate-wild-do-table=name</code>	
<b>Option-File Format</b>	<code>replicate-wild-do-table</code>	
	<b>Permitted Values</b>	
	<b>Type</b>	string

Tells the slave thread to restrict replication to statements where any of the updated tables match the specified database and table name patterns. Patterns can contain the “%” and “\_” wildcard characters, which have the same meaning as for the `LIKE` pattern-matching operator. To specify more than one table, use this option multiple times, once for each table. This works for cross-database updates. See [How Servers Evaluate Replication Filtering Rules](#).

This option applies to tables, views, and triggers. It does not apply to stored procedures and functions. To filter statements operating on the latter objects, use one or more of the `--replicate-*-db` options.

Example: `--replicate-wild-do-table=foo%.bar%` [30] replicates only updates that use a table where the database name starts with `foo` and the table name starts with `bar`.

If the table name pattern is `%`, it matches any table name and the option also applies to database-level statements (`CREATE DATABASE`, `DROP DATABASE`, and `ALTER DATABASE`). For example, if you use `--replicate-wild-do-table=foo%.%` [30], database-level statements are replicated if the database name matches the pattern `foo%`.

To include literal wildcard characters in the database or table name patterns, escape them with a backslash. For example, to replicate all tables of a database that is named `my_own%db`, but not replicate tables from the `my1ownAABCdb` database, you should escape the “`_`” and “`%`” characters like this: `--replicate-wild-do-table=my\_own\%db` [30]. If you use the option on the command line, you might need to double the backslashes or quote the option value, depending on your command interpreter. For example, with the `bash` shell, you would need to type `--replicate-wild-do-table=my\\\_own\\%db` [30].

- `--replicate-wild-ignore-table=db_name.tbl_name` [31]

<b>Command-Line Format</b>	<code>--replicate-wild-ignore-table=name</code>	
<b>Option-File Format</b>	<code>replicate-wild-ignore-table</code>	
	<b>Permitted Values</b>	
	<b>Type</b>	<code>string</code>

Tells the slave thread not to replicate a statement where any table matches the given wildcard pattern. To specify more than one table to ignore, use this option multiple times, once for each table. This works for cross-database updates. See [How Servers Evaluate Replication Filtering Rules](#).

Example: `--replicate-wild-ignore-table=foo%.bar%` [31] does not replicate updates that use a table where the database name starts with `foo` and the table name starts with `bar`.

For information about how matching works, see the description of the `--replicate-wild-do-table` [30] option. The rules for including literal wildcard characters in the option value are the same as for `--replicate-wild-ignore-table` [31] as well.

- `--report-host=host_name` [31]

<b>Command-Line Format</b>	<code>--report-host=host_name</code>	
<b>Option-File Format</b>	<code>report-host</code>	
<b>Option Sets Variable</b>	Yes, <code>report_host</code>	
<b>Variable Name</b>	<code>report-host</code>	
<b>Variable Scope</b>	Global	
<b>Dynamic Variable</b>	No	
	<b>Permitted Values</b>	
	<b>Type</b>	<code>string</code>

The host name or IP address of the slave to be reported to the master during slave registration. This value appears in the output of `SHOW SLAVE HOSTS` on the master server. Leave the value unset if you do not want the slave to register itself with the master. Note that it is not sufficient for the master to simply read the IP address of the slave from the TCP/IP socket after the slave connects. Due to NAT

and other routing issues, that IP may not be valid for connecting to the slave from the master or other hosts.

- `--report-password=password` [32]

<b>Command-Line Format</b>	<code>--report-password=name</code>	
<b>Option-File Format</b>	<code>report-password</code>	
<b>Option Sets Variable</b>	Yes, <code>report_password</code>	
<b>Variable Name</b>	<code>report-password</code>	
<b>Variable Scope</b>	Global	
<b>Dynamic Variable</b>	No	
	<b>Permitted Values</b>	
	<b>Type</b>	<code>string</code>

The account password of the slave to be reported to the master during slave registration. This value appears in the output of `SHOW SLAVE HOSTS` on the master server if the `--show-slave-auth-info` [33] option is given.

Although the name of this option might imply otherwise, `--report-password` is not connected to the MySQL user privilege system and so is not necessarily (or even likely to be) the same as the password for the MySQL replication user account.

- `--report-port=slave_port_num` [32]

<b>Command-Line Format</b>	<code>--report-port=#</code>	
<b>Option-File Format</b>	<code>report-port</code>	
<b>Option Sets Variable</b>	Yes, <code>report_port</code>	
<b>Variable Name</b>	<code>report-port</code>	
<b>Variable Scope</b>	Global	
<b>Dynamic Variable</b>	No	
	<b>Permitted Values</b>	
	<b>Type</b>	<code>numeric</code>
	<b>Default</b>	<code>3306</code>

The TCP/IP port number for connecting to the slave, to be reported to the master during slave registration. Set this only if the slave is listening on a nondefault port or if you have a special tunnel from the master or other clients to the slave. If you are not sure, do not use this option.

- `--report-user=user_name` [32]

<b>Command-Line Format</b>	<code>--report-user=name</code>	
<b>Option-File Format</b>	<code>report-user</code>	
<b>Option Sets Variable</b>	Yes, <code>report_user</code>	
<b>Variable Name</b>	<code>report-user</code>	
<b>Variable Scope</b>	Global	
<b>Dynamic Variable</b>	No	
	<b>Permitted Values</b>	

	<b>Type</b>	string
--	-------------	--------

The account user name of the slave to be reported to the master during slave registration. This value appears in the output of `SHOW SLAVE HOSTS` on the master server if the `--show-slave-auth-info` [33] option is given.

Although the name of this option might imply otherwise, `--report-user` is not connected to the MySQL user privilege system and so is not necessarily (or even likely to be) the same as the name of the MySQL replication user account.

- `--show-slave-auth-info` [33]

<b>Command-Line Format</b>	<code>--show-slave-auth-info</code>	
<b>Option-File Format</b>	<code>show-slave-auth-info</code>	
	<b>Permitted Values</b>	
	<b>Type</b>	boolean
	<b>Default</b>	FALSE

Display slave user names and passwords in the output of `SHOW SLAVE HOSTS` on the master server for slaves started with the `--report-user` [32] and `--report-password` [32] options.

- `--skip-slave-start` [33]

<b>Command-Line Format</b>	<code>--skip-slave-start</code>	
<b>Option-File Format</b>	<code>skip-slave-start</code>	
	<b>Permitted Values</b>	
	<b>Type</b>	boolean
	<b>Default</b>	FALSE

Tells the slave server not to start the slave threads when the server starts. To start the threads later, use a `START SLAVE` statement.

- `--slave_compressed_protocol={0|1}` [36]

<b>Command-Line Format</b>	<code>--slave_compressed_protocol</code>	
<b>Option-File Format</b>	<code>slave_compressed_protocol</code>	
<b>Option Sets Variable</b>	Yes, <code>slave_compressed_protocol</code> [36]	
<b>Variable Name</b>	<code>slave_compressed_protocol</code>	
<b>Variable Scope</b>	Global	
<b>Dynamic Variable</b>	Yes	
	<b>Permitted Values</b>	
	<b>Type</b>	boolean
	<b>Default</b>	OFF

If this option is set to 1, use compression for the slave/master protocol if both the slave and the master support it. The default is 0 (no compression).

- `--slave-load-tmpdir=file_name` [33]33

The name of the directory where the slave creates temporary files. This option is by default equal to the value of the `tmpdir` system variable. When the slave SQL thread replicates a `LOAD DATA INFILE` statement, it extracts the file to be loaded from the relay log into temporary files, and then loads these into the table. If the file loaded on the master is huge, the temporary files on the slave are huge, too. Therefore, it might be advisable to use this option to tell the slave to put temporary files in a directory located in some file system that has a lot of available space. In that case, the relay logs are huge as well, so you might also want to use the `--relay-log` [26] option to place the relay logs in that file system.

The directory specified by this option should be located in a disk-based file system (not a memory-based file system) because the temporary files used to replicate `LOAD DATA INFILE` must survive machine restarts. The directory also should not be one that is cleared by the operating system during the system startup process.

- `--slave-net-timeout=seconds` [34]

<b>Command-Line Format</b>	<code>--slave-net-timeout=#</code>	
<b>Option-File Format</b>	<code>slave-net-timeout</code>	
<b>Option Sets Variable</b>	Yes, <code>slave_net_timeout</code> [37]	
<b>Variable Name</b>	<code>slave_net_timeout</code>	
<b>Variable Scope</b>	Global	
<b>Dynamic Variable</b>	Yes	
	<b>Permitted Values</b>	
	<b>Type</b>	<code>numeric</code>
	<b>Default</b>	<code>3600</code>
	<b>Min Value</b>	<code>1</code>

The number of seconds to wait for more data from the master before the slave considers the connection broken, aborts the read, and tries to reconnect. The first retry occurs immediately after the timeout. The interval between retries is controlled by the `CHANGE MASTER TO` statement or `--master-connect-retry` [23] option and the number of reconnection attempts is limited by the `--master-retry-count` [25] option. The default is 3600 seconds (one hour).

- `--slave-skip-errors=[err_code1,err_code2,...|all]` [34]

<b>Command-Line Format</b>	<code>--slave-skip-errors=name</code>	
<b>Option-File Format</b>	<code>slave-skip-errors</code>	
<b>Option Sets Variable</b>	Yes, <code>slave_skip_errors</code> [37]	
<b>Variable Name</b>	<code>slave_skip_errors</code>	
<b>Variable Scope</b>	Global	
<b>Dynamic Variable</b>	No	
	<b>Permitted Values</b>	
	<b>Type</b>	<code>string</code>
	<b>Default</b>	<code>OFF</code>
	<b>Valid Values</b>	<code>[list of error codes]</code> <code>all</code>



Normally, replication stops when an error occurs on the slave. This gives you the opportunity to resolve the inconsistency in the data manually. This option tells the slave SQL thread to continue replication when a statement returns any of the errors listed in the option value.

Do not use this option unless you fully understand why you are getting errors. If there are no bugs in your replication setup and client programs, and no bugs in MySQL itself, an error that stops replication should never occur. Indiscriminate use of this option results in slaves becoming hopelessly out of synchrony with the master, with you having no idea why this has occurred.

For error codes, you should use the numbers provided by the error message in your slave error log and in the output of `SHOW SLAVE STATUS`. [Errors, Error Codes, and Common Problems](#), lists server error codes.

You can also (but should not) use the very nonrecommended value of `all` to cause the slave to ignore all error messages and keeps going regardless of what happens. Needless to say, if you use `all`, there are no guarantees regarding the integrity of your data. Please do not complain (or file bug reports) in this case if the slave's data is not anywhere close to what it is on the master. *You have been warned.*

Examples:

```
--slave-skip-errors=1062,1053
--slave-skip-errors=all
```

**System variables used on replication slaves.** The following list describes system variables for controlling replication slave servers. They can be set at server startup and some of them can be changed at runtime using `SET`. Server options used with replication slaves are listed earlier in this section.

- `init_slave` [35]

<b>Command-Line Format</b>	<code>--init-slave=name</code>	
<b>Option-File Format</b>	<code>init_slave</code>	
<b>Option Sets Variable</b>	Yes, <code>init_slave</code> [35]	
<b>Variable Name</b>	<code>init_slave</code>	
<b>Variable Scope</b>	Global	
<b>Dynamic Variable</b>	Yes	
	<b>Permitted Values</b>	
	<b>Type</b>	<code>string</code>

This variable is similar to `init_connect`, but is a string to be executed by a slave server each time the SQL thread starts. The format of the string is the same as for the `init_connect` variable.

#### Note

The SQL thread sends an acknowledgment to the client before it executes `init_slave` [35]. Therefore, it is not guaranteed that `init_slave` [35] has been executed when `START SLAVE` returns. See [START SLAVE Syntax](#), for more information.

- `relay_log` [35]

<b>Command-Line Format</b>	<code>--relay-log=name</code>
<b>Option-File Format</b>	<code>relay-log</code>

<b>Option Sets Variable</b>	Yes, <code>relay_log</code> [35]	
<b>Variable Name</b>	<code>relay-log</code>	
<b>Variable Scope</b>	Global	
<b>Dynamic Variable</b>	No	
	<b>Permitted Values</b>	
	<b>Type</b>	<code>file name</code>

The name of the relay log file.

- `relay_log_index` [36]

<b>Command-Line Format</b>	<code>--relay-log-index</code>	
<b>Option-File Format</b>	<code>relay_log_index</code>	
<b>Option Sets Variable</b>	Yes, <code>relay_log_index</code> [36]	
<b>Variable Name</b>	<code>relay_log_index</code>	
<b>Variable Scope</b>	Global	
<b>Dynamic Variable</b>	No	
	<b>Permitted Values</b>	
	<b>Type</b>	<code>string</code>
	<b>Default</b>	<code>*host_name*-relay-bin.index</code>

The name of the relay log index file. The default name is `host_name-relay-bin.index` in the data directory, where `host_name` is the name of the slave server.

- `relay_log_info_file` [36]

<b>Command-Line Format</b>	<code>--relay-log-info-file=file_name</code>	
<b>Option-File Format</b>	<code>relay_log_info_file</code>	
<b>Option Sets Variable</b>	Yes, <code>relay_log_info_file</code> [36]	
<b>Variable Name</b>	<code>relay_log_info_file</code>	
<b>Variable Scope</b>	Global	
<b>Dynamic Variable</b>	No	
	<b>Permitted Values</b>	
	<b>Type</b>	<code>string</code>
	<b>Default</b>	<code>relay-log.info</code>

The name of the file in which the slave records information about the relay logs. The default name is `relay-log.info` in the data directory.

- `rpl_recovery_rank` [36]

This variable is unused, and is removed in MySQL 5.6.

- `slave_compressed_protocol` [36]

<b>Command-Line Format</b>	<code>--slave_compressed_protocol</code>
----------------------------	--

<b>Option-File Format</b>	<code>slave_compressed_protocol</code>	
<b>Option Sets Variable</b>	Yes, <code>slave_compressed_protocol</code> [36]	
<b>Variable Name</b>	<code>slave_compressed_protocol</code>	
<b>Variable Scope</b>	Global	
<b>Dynamic Variable</b>	Yes	
	<b>Permitted Values</b>	
	<b>Type</b>	<code>boolean</code>
	<b>Default</b>	<code>OFF</code>

Whether to use compression of the slave/master protocol if both the slave and the master support it.

- `slave_load_tmpdir` [37]

<b>Command-Line Format</b>	<code>--slave-load-tmpdir=path</code>	
<b>Option-File Format</b>	<code>slave-load-tmpdir</code>	
<b>Option Sets Variable</b>	Yes, <code>slave_load_tmpdir</code> [37]	
<b>Variable Name</b>	<code>slave_load_tmpdir</code>	
<b>Variable Scope</b>	Global	
<b>Dynamic Variable</b>	No	
	<b>Permitted Values</b>	
	<b>Type</b>	<code>file name</code>
	<b>Default</b>	<code>/tmp</code>

The name of the directory where the slave creates temporary files for replicating `LOAD DATA INFILE` statements.

- `slave_net_timeout` [37]

<b>Command-Line Format</b>	<code>--slave-net-timeout=#</code>	
<b>Option-File Format</b>	<code>slave-net-timeout</code>	
<b>Option Sets Variable</b>	Yes, <code>slave_net_timeout</code> [37]	
<b>Variable Name</b>	<code>slave_net_timeout</code>	
<b>Variable Scope</b>	Global	
<b>Dynamic Variable</b>	Yes	
	<b>Permitted Values</b>	
	<b>Type</b>	<code>numeric</code>
	<b>Default</b>	<code>3600</code>
	<b>Min Value</b>	<code>1</code>

The number of seconds to wait for more data from a master/slave connection before aborting the read. This timeout applies only to TCP/IP connections, not to connections made using Unix socket files, named pipes, or shared memory.

- `slave_skip_errors` [37]

<b>Command-Line Format</b>	<code>--slave-skip-errors=name</code>	
<b>Option-File Format</b>	<code>slave-skip-errors</code>	
<b>Option Sets Variable</b>	Yes, <code>slave_skip_errors</code> [37]	
<b>Variable Name</b>	<code>slave_skip_errors</code>	
<b>Variable Scope</b>	Global	
<b>Dynamic Variable</b>	No	
	<b>Permitted Values</b>	
	<b>Type</b>	<code>string</code>
	<b>Default</b>	<code>OFF</code>
	<b>Valid Values</b>	<code>[list of error codes]</code> <code>all</code>

Normally, replication stops when an error occurs on the slave. This gives you the opportunity to resolve the inconsistency in the data manually. This variable tells the slave SQL thread to continue replication when a statement returns any of the errors listed in the variable value.

- `slave_transaction_retries` [38]

<b>Version Introduced</b>	5.0.3	
<b>Command-Line Format</b>	<code>--slave_transaction_retries=#</code>	
<b>Option-File Format</b>	<code>slave_transaction_retries</code>	
<b>Option Sets Variable</b>	Yes, <code>slave_transaction_retries</code> [38]	
<b>Variable Name</b>	<code>slave_transaction_retries</code>	
<b>Variable Scope</b>	Global	
<b>Dynamic Variable</b>	Yes	
	<b>Permitted Values</b>	
	<b>Platform Bit Size</b>	32
	<b>Type</b>	<code>numeric</code>
	<b>Default</b>	10
	<b>Range</b>	0 .. 4294967295
	<b>Permitted Values</b>	
	<b>Platform Bit Size</b>	64
	<b>Type</b>	<code>numeric</code>
	<b>Default</b>	10
	<b>Range</b>	0 .. 18446744073709547520

If a replication slave SQL thread fails to execute a transaction because of an InnoDB deadlock or because the transaction's execution time exceeded InnoDB's `innodb_lock_wait_timeout` or NDBCLUSTER's `TransactionDeadlockDetectionTimeout` or `TransactionInactiveTimeout`, it automatically retries `slave_transaction_retries` [38] times before stopping with an error. Prior

to MySQL 5.0.3, the default is 0, and you must explicitly set the value greater than 0 to enable the “retry” behavior. In MySQL 5.0.3 or newer, the default is 10.

- `sql_slave_skip_counter` [39]

<b>Variable Name</b>	<code>sql_slave_skip_counter</code>	
<b>Variable Scope</b>	Global	
<b>Dynamic Variable</b>	Yes	
	<b>Permitted Values</b>	
	<b>Type</b>	<code>numeric</code>

The number of events from the master that a slave server should skip.

### Important

If skipping the number of events specified by setting this variable would cause the slave to begin in the middle of an event group, the slave continues to skip until it finds the beginning of the next event group and begins from that point. For more information, see [SET GLOBAL sql\\_slave\\_skip\\_counter Syntax](#).

## Binary Log Options and Variables

You can use the `mysqld` options and system variables that are described in this section to affect the operation of the binary log as well as to control which statements are written to the binary log. For additional information about the binary log, see [The Binary Log](#). For additional information about using MySQL server options and system variables, see [Server Command Options](#), and [Server System Variables](#).

**Startup options used with binary logging.** The following list describes startup options for enabling and configuring the binary log. System variables used with binary logging are discussed later in this section.

- `--log-bin[=base_name]` [39]

<b>Command-Line Format</b>	<code>--log-bin</code>	
<b>Option-File Format</b>	<code>log-bin</code>	
<b>Variable Name</b>	<code>log_bin</code>	
<b>Variable Scope</b>	Global	
<b>Dynamic Variable</b>	No	
	<b>Permitted Values</b>	
	<b>Type</b>	<code>file name</code>
	<b>Default</b>	<code>OFF</code>

Enable binary logging. The server logs all statements that change data to the binary log, which is used for backup and replication. See [The Binary Log](#).

The option value, if given, is the basename for the log sequence. The server creates binary log files in sequence by adding a numeric suffix to the basename. It is recommended that you specify a basename (see [Known Issues in MySQL](#), for the reason). Otherwise, MySQL uses `host_name-bin` as the basename.

Setting this option causes the `log_bin` system variable to be set to `ON` (or `1`), and not to the basename. This is a known issue; see Bug #19614 for more information.

- `--log-bin-index[=file_name]` [40]

<b>Command-Line Format</b>	<code>--log-bin-index=name</code>	
<b>Option-File Format</b>	<code>log-bin-index</code>	
	<b>Permitted Values</b>	
	<b>Type</b>	<code>file name</code>
	<b>Default</b>	<code>OFF</code>

The index file for binary log file names. See [The Binary Log](#). If you omit the file name, and if you did not specify one with `--log-bin` [39], MySQL uses `host_name-bin.index` as the file name.

- `--log-bin-trust-function-creators[={0|1}]` [40]

<b>Version Introduced</b>	5.0.16	
<b>Command-Line Format</b>	<code>--log-bin-trust-function-creators</code>	
<b>Option-File Format</b>	<code>log-bin-trust-function-creators</code>	
<b>Option Sets Variable</b>	Yes, <code>log_bin_trust_function_creators</code>	
<b>Variable Name</b>	<code>log_bin_trust_function_creators</code>	
<b>Variable Scope</b>	Global	
<b>Dynamic Variable</b>	Yes	
	<b>Permitted Values</b>	
	<b>Type</b>	<code>boolean</code>
	<b>Default</b>	<code>FALSE</code>

This option sets the corresponding `log_bin_trust_function_creators` system variable. If no argument is given, the option sets the variable to 1. `log_bin_trust_function_creators` affects how MySQL enforces restrictions on stored function and trigger creation. See [Binary Logging of Stored Programs](#).

This option was added in MySQL 5.0.16.

- `--log-bin-trust-routine-creators[={0|1}]` [40]

<b>Version Introduced</b>	5.0.6
<b>Version Deprecated</b>	5.0.16
<b>Command-Line Format</b>	<code>--log-bin-trust-routine-creators</code>
<b>Option-File Format</b>	<code>log-bin-trust-routine-creators</code>
<b>Option Sets Variable</b>	Yes, <code>log_bin_trust_routine_creators</code>
<b>Variable Name</b>	<code>log-bin-trust-routine-creators</code>
<b>Variable Scope</b>	Global
<b>Dynamic Variable</b>	Yes
<b>Deprecated</b>	5.0.16, by <code>log-bin-trust-function-creators</code>

Permitted Values	
Type	boolean
Default	FALSE

This is the old name for `--log-bin-trust-function-creators` [40]. Before MySQL 5.0.16, it also applies to stored procedures, not just stored functions and sets the `log_bin_trust_routine_creators` system variable. As of 5.0.16, this option is deprecated. It is recognized for backward compatibility but its use results in a warning.

This option was added in MySQL 5.0.6.

**Statement selection options.** The options in the following list affect which statements are written to the binary log, and thus sent by a replication master server to its slaves. There are also options for slave servers that control which statements received from the master should be executed or ignored. For details, see [Replication Slave Options and Variables](#).

- `--binlog-do-db=db_name` [41]

Command-Line Format	<code>--binlog-do-db=name</code>	
Option-File Format	<code>binlog-do-db</code>	
	Permitted Values	
	Type	<code>string</code>

This option affects binary logging in a manner similar to the way that `--replicate-do-db` [28] affects replication.

Tell the server to restrict binary logging to updates for which the default database is `db_name` (that is, the database selected by `USE`). All other databases that are not explicitly mentioned are ignored. If you use this option, you should ensure that you do updates only in the default database.

There is an exception to this for `CREATE DATABASE`, `ALTER DATABASE`, and `DROP DATABASE` statements. The server uses the database named in the statement (not the default database) to decide whether it should log the statement.

An example of what does not work as you might expect: If the server is started with `--binlog-do-db=sales` [41] and you issue the following statements, the `UPDATE` statement is *not* logged:

```
USE prices;
UPDATE sales.january SET amount=amount+1000;
```

The main reason for this “just check the default database” behavior is that it is difficult from the statement alone to know whether it should be replicated (for example, if you are using multiple-table `DELETE` statements or multiple-table `UPDATE` statements that act across multiple databases). It is also faster to check only the default database rather than all databases if there is no need.

Another case which may not be self-evident occurs when a given database is replicated even though it was not specified when setting the option. If the server is started with `--binlog-do-db=sales`, the following `UPDATE` statement is logged even though `prices` was not included when setting `--binlog-do-db`:

```
USE sales;
UPDATE prices.discounts SET percentage = percentage + 10;
```

Because `sales` is the default database when the `UPDATE` statement is issued, the `UPDATE` is logged.

**Important**

To log multiple databases, use this option multiple times, specifying the option once for each database to be logged. Because database names can contain commas, the list will be treated as the name of a single database if you supply a comma-separated list.

- `--binlog-ignore-db=db_name` [42]

<b>Command-Line Format</b>	<code>--binlog-ignore-db=name</code>	
<b>Option-File Format</b>	<code>binlog-ignore-db</code>	
	<b>Permitted Values</b>	
	<b>Type</b>	<code>string</code>

This option affects binary logging in a manner similar to the way that `--replicate-ignore-db` [28] affects replication.

Tell the server to suppress binary logging of updates for which the default database is `db_name` (that is, the database selected by `USE`). If you use this option, you should ensure that you do updates only in the default database.

As with the `--binlog-do-db` [41] option, there is an exception for the `CREATE DATABASE`, `ALTER DATABASE`, and `DROP DATABASE` statements. The server uses the database named in the statement (not the default database) to decide whether it should log the statement.

An example of what does not work as you might expect: If the server is started with `binlog-ignore-db=sales`, and you run `USE prices; UPDATE sales.january SET amount = amount + 1000;`, this statement *is* written into the binary log.

**Important**

To ignore multiple databases, use this option multiple times, specifying the option once for each database to be ignored. Because database names can contain commas, the list will be treated as the name of a single database if you supply a comma-separated list.

**Testing and debugging options.** The following binary log options are used in replication testing and debugging. They are not intended for use in normal operations.

- `--max-binlog-dump-events=N` [42]

<b>Command-Line Format</b>	<code>--max-binlog-dump-events=#</code>	
<b>Option-File Format</b>	<code>max-binlog-dump-events</code>	
	<b>Permitted Values</b>	
	<b>Type</b>	<code>numeric</code>
	<b>Default</b>	<code>0</code>

This option is used internally by the MySQL test suite for replication testing and debugging.

- `--sporadic-binlog-dump-fail` [42]

<b>Command-Line Format</b>	<code>--sporadic-binlog-dump-fail</code>
----------------------------	--



<b>Option-File Format</b>	<code>sporadic-binlog-dump-fail</code>	
	<b>Permitted Values</b>	
	<b>Type</b>	<code>boolean</code>
	<b>Default</b>	<code>FALSE</code>

This option is used internally by the MySQL test suite for replication testing and debugging.

**System variables used with the binary log.** The following list describes system variables for controlling binary logging. They can be set at server startup and some of them can be changed at runtime using `SET`. Server options used to control binary logging are listed earlier in this section.

- `log_slave_updates` [43]

Whether updates received by a slave server from a master server should be logged to the slave's own binary log. Binary logging must be enabled on the slave for this variable to have any effect. See [Replication Slave Options and Variables](#).

- `max_binlog_cache_size` [43]

<b>Command-Line Format</b>	<code>--max_binlog_cache_size=#</code>
<b>Option-File Format</b>	<code>max_binlog_cache_size</code>
<b>Option Sets Variable</b>	Yes, <code>max_binlog_cache_size</code> [43]
<b>Variable Name</b>	<code>max_binlog_cache_size</code>
<b>Variable Scope</b>	Global
<b>Dynamic Variable</b>	Yes

If a multiple-statement transaction requires more than this many bytes of memory, the server generates a `Multi-statement transaction required more than 'max_binlog_cache_size' bytes of storage` error. The minimum value is 4096. The maximum and default values are 4GB on 32-bit platforms and 16PB (petabytes) on 64-bit platforms.

In MySQL 5.0, a change in `max_binlog_cache_size` [43] takes immediate effect for all active sessions.

- `max_binlog_size` [43]

<b>Command-Line Format</b>	<code>--max_binlog_size=#</code>	
<b>Option-File Format</b>	<code>max_binlog_size</code>	
<b>Option Sets Variable</b>	Yes, <code>max_binlog_size</code> [43]	
<b>Variable Name</b>	<code>max_binlog_size</code>	
<b>Variable Scope</b>	Global	
<b>Dynamic Variable</b>	Yes	
	<b>Permitted Values</b>	
	<b>Type</b>	<code>numeric</code>
	<b>Default</b>	<code>1073741824</code>
	<b>Range</b>	<code>4096 .. 1073741824</code>

If a write to the binary log causes the current log file size to exceed the value of this variable, the server rotates the binary logs (closes the current file and opens the next one). The minimum value is 4096 bytes. The maximum and default value is 1GB.

A transaction is written in one chunk to the binary log, so it is never split between several binary logs. Therefore, if you have big transactions, you might see binary log files larger than `max_binlog_size` [43].

If `max_relay_log_size` is 0, the value of `max_binlog_size` [43] applies to relay logs as well.

- `sync_binlog` [44]

<b>Version Introduced</b>	5.0.1	
<b>Command-Line Format</b>	<code>--sync-binlog=#</code>	
<b>Option-File Format</b>	<code>sync_binlog</code>	
<b>Option Sets Variable</b>	Yes, <code>sync_binlog</code> [44]	
<b>Variable Name</b>	<code>sync_binlog</code>	
<b>Variable Scope</b>	Global	
<b>Dynamic Variable</b>	Yes	
	<b>Permitted Values</b>	
	<b>Platform Bit Size</b>	32
	<b>Type</b>	numeric
	<b>Default</b>	0
	<b>Range</b>	0 .. 4294967295
	<b>Permitted Values</b>	
	<b>Platform Bit Size</b>	64
	<b>Type</b>	numeric
	<b>Default</b>	0
	<b>Range</b>	0 .. 18446744073709547520

If the value of this variable is greater than 0, the MySQL server synchronizes its binary log to disk (using `fdatsync()`) after every `sync_binlog` [44] writes to the binary log. There is one write to the binary log per statement if autocommit is enabled, and one write per transaction otherwise. The default value of `sync_binlog` [44] is 0, which does no synchronizing to disk. A value of 1 is the safest choice because in the event of a crash you lose at most one statement or transaction from the binary log. However, it is also the slowest choice (unless the disk has a battery-backed cache, which makes synchronization very fast).

If the value of `sync_binlog` [44] is 0 (the default), no extra flushing is done. The server relies on the operating system to flush the file contents occasionally as for any other file.

## Common Replication Administration Tasks

Once replication has been started it should execute without requiring much regular administration. Depending on your replication environment, you will want to check the replication status of each slave periodically, daily, or even more frequently.

## Checking Replication Status

The most common task when managing a replication process is to ensure that replication is taking place and that there have been no errors between the slave and the master. The primary statement for this is `SHOW SLAVE STATUS`, which you must execute on each slave:

```
mysql> SHOW SLAVE STATUS\G
***** 1. row *****
      Slave_IO_State: Waiting for master to send event
        Master_Host: master1
        Master_User: root
        Master_Port: 3306
        Connect_Retry: 60
        Master_Log_File: mysql-bin.000004
        Read_Master_Log_Pos: 931
        Relay_Log_File: slave1-relay-bin.000056
        Relay_Log_Pos: 950
        Relay_Master_Log_File: mysql-bin.000004
        Slave_IO_Running: Yes
        Slave_SQL_Running: Yes
        Replicate_Do_DB:
        Replicate_Ignore_DB:
        Replicate_Do_Table:
        Replicate_Ignore_Table:
        Replicate_Wild_Do_Table:
        Replicate_Wild_Ignore_Table:
          Last_Errno: 0
          Last_Error:
        Skip_Counter: 0
        Exec_Master_Log_Pos: 931
        Relay_Log_Space: 1365
        Until_Condition: None
        Until_Log_File:
        Until_Log_Pos: 0
        Master_SSL_Allowed: No
        Master_SSL_CA_File:
        Master_SSL_CA_Path:
        Master_SSL_Cert:
        Master_SSL_Cipher:
        Master_SSL_Key:
        Seconds_Behind_Master: 0
```

The key fields from the status report to examine are:

- **Slave\_IO\_State:** The current status of the slave. See [Replication Slave I/O Thread States](#), and [Replication Slave SQL Thread States](#), for more information.
- **Slave\_IO\_Running:** Whether the I/O thread for reading the master's binary log is running. Normally, you want this to be **Yes** unless you have not yet started replication or have explicitly stopped it with `STOP SLAVE`.
- **Slave\_SQL\_Running:** Whether the SQL thread for executing events in the relay log is running. As with the I/O thread, this should normally be **Yes**.
- **Last\_Error:** The last error registered when processing the relay log. Ideally this should be blank, indicating no error.
- **Seconds\_Behind\_Master:** The number of seconds that the slave SQL thread is behind processing the master binary log. A high number (or an increasing one) can indicate that the slave is unable to handle events from the master in a timely fashion.

A value of 0 for **Seconds\_Behind\_Master** can usually be interpreted as meaning that the slave has caught up with the master, but there are some cases where this is not strictly true. For example, this can

occur if the network connection between master and slave is broken but the slave I/O thread has not yet noticed this—that is, `slave_net_timeout` [37] has not yet elapsed.

It is also possible that transient values for `Seconds_Behind_Master` may not reflect the situation accurately. When the slave SQL thread has caught up on I/O, `Seconds_Behind_Master` displays 0; but when the slave I/O thread is still queuing up a new event, `Seconds_Behind_Master` may show a large value until the SQL thread finishes executing the new event. This is especially likely when the events have old timestamps; in such cases, if you execute `SHOW SLAVE STATUS` several times in a relatively short period, you may see this value change back and forth repeatedly between 0 and a relatively large value.

Several pairs of fields provide information about the progress of the slave in reading events from the master binary log and processing them in the relay log:

- (`Master_Log_File`, `Read_Master_Log_Pos`): Coordinates in the master binary log indicating how far the slave I/O thread has read events from that log.
- (`Relay_Master_Log_File`, `Exec_Master_Log_Pos`): Coordinates in the master binary log indicating how far the slave SQL thread has executed events received from that log.
- (`Relay_Log_File`, `Relay_Log_Pos`): Coordinates in the slave relay log indicating how far the slave SQL thread has executed the relay log. These correspond to the preceding coordinates, but are expressed in slave relay log coordinates rather than master binary log coordinates.

On the master, you can check the status of connected slaves using `SHOW PROCESSLIST` to examine the list of running processes. Slave connections have `Binlog Dump` in the `Command` field:

```
mysql> SHOW PROCESSLIST \G;
***** 4. row *****
      Id: 10
     User: root
    Host: slave1:58371
       db: NULL
Command: Binlog Dump
      Time: 777
    State: Has sent all binlog to slave; waiting for binlog to be updated
     Info: NULL
```

Because it is the slave that drives the replication process, very little information is available in this report.

For slaves that were started with the `--report-host` [31] option and are connected to the master, the `SHOW SLAVE HOSTS` statement on the master shows basic information about the slaves. The output includes the ID of the slave server, the value of the `--report-host` [31] option, the connecting port, and master ID:

```
mysql> SHOW SLAVE HOSTS;
+-----+-----+-----+-----+-----+
| Server_id | Host   | Port | Rpl_recovery_rank | Master_id |
+-----+-----+-----+-----+-----+
|         10 | slave1 | 3306 |                  0 |          1 |
+-----+-----+-----+-----+-----+
1 row in set (0.00 sec)
```

## Pausing Replication on the Slave

You can stop and start the replication of statements on the slave using the `STOP SLAVE` and `START SLAVE` statements.

To stop processing of the binary log from the master, use `STOP SLAVE`:

```
mysql> STOP SLAVE;
```

When replication is stopped, the slave I/O thread stops reading events from the master binary log and writing them to the relay log, and the SQL thread stops reading events from the relay log and executing them. You can pause the I/O or SQL thread individually by specifying the thread type:

```
mysql> STOP SLAVE IO_THREAD;  
mysql> STOP SLAVE SQL_THREAD;
```

To start execution again, use the `START SLAVE` statement:

```
mysql> START SLAVE;
```

To start a particular thread, specify the thread type:

```
mysql> START SLAVE IO_THREAD;  
mysql> START SLAVE SQL_THREAD;
```

For a slave that performs updates only by processing events from the master, stopping only the SQL thread can be useful if you want to perform a backup or other task. The I/O thread will continue to read events from the master but they are not executed. This makes it easier for the slave to catch up when you restart the SQL thread.

Stopping only the I/O thread enables the events in the relay log to be executed by the SQL thread up to the point where the relay log ends. This can be useful when you want to pause execution to catch up with events already received from the master, when you want to perform administration on the slave but also ensure that it has processed all updates to a specific point. This method can also be used to pause event receipt on the slave while you conduct administration on the master. Stopping the I/O thread but permitting the SQL thread to run helps ensure that there is not a massive backlog of events to be executed when replication is started again.



---

## Chapter 3. Replication Solutions

### Table of Contents

Using Replication for Backups .....	49
Backing Up a Slave Using <code>mysqldump</code> .....	50
Backing Up Raw Data from a Slave .....	50
Using Replication with Different Master and Slave Storage Engines .....	51
Using Replication for Scale-Out .....	52
Replicating Different Databases to Different Slaves .....	54
Improving Replication Performance .....	55
Switching Masters During Failover .....	56
Setting Up Replication Using SSL .....	59

Replication can be used in many different environments for a range of purposes. This section provides general notes and advice on using replication for specific solution types.

For information on using replication in a backup environment, including notes on the setup, backup procedure, and files to back up, see [Using Replication for Backups](#).

For advice and tips on using different storage engines on the master and slaves, see [Using Replication with Different Master and Slave Storage Engines](#).

Using replication as a scale-out solution requires some changes in the logic and operation of applications that use the solution. See [Using Replication for Scale-Out](#).

For performance or data distribution reasons, you may want to replicate different databases to different replication slaves. See [Replicating Different Databases to Different Slaves](#)

As the number of replication slaves increases, the load on the master can increase and lead to reduced performance (because of the need to replicate the binary log to each slave). For tips on improving your replication performance, including using a single secondary server as an replication master, see [Improving Replication Performance](#).

For guidance on switching masters, or converting slaves into masters as part of an emergency failover solution, see [Switching Masters During Failover](#).

To secure your replication communication, you can use SSL to encrypt the communication channel. For step-by-step instructions, see [Setting Up Replication Using SSL](#).

### Using Replication for Backups

To use replication as a backup solution, replicate data from the master to a slave, and then back up the data slave. The slave can be paused and shut down without affecting the running operation of the master, so you can produce an effective snapshot of “live” data that would otherwise require the master to be shut down.

How you back up a database depends on its size and whether you are backing up only the data, or the data and the replication slave state so that you can rebuild the slave in the event of failure. There are therefore two choices:

- If you are using replication as a solution to enable you to back up the data on the master, and the size of your database is not too large, the `mysqldump` tool may be suitable. See [Backing Up a Slave Using `mysqldump`](#).

- For larger databases, where `mysqldump` would be impractical or inefficient, you can back up the raw data files instead. Using the raw data files option also means that you can back up the binary and relay logs that will enable you to recreate the slave in the event of a slave failure. For more information, see [Backing Up Raw Data from a Slave](#).

## Backing Up a Slave Using `mysqldump`

Using `mysqldump` to create a copy of a database enables you to capture all of the data in the database in a format that enables the information to be imported into another instance of MySQL Server (see [mysqldump — A Database Backup Program](#)). Because the format of the information is SQL statements, the file can easily be distributed and applied to running servers in the event that you need access to the data in an emergency. However, if the size of your data set is very large, `mysqldump` may be impractical.

When using `mysqldump`, you should stop replication on the slave before starting the dump process to ensure that the dump contains a consistent set of data:

1. Stop the slave from processing requests. You can stop replication completely on the slave using `mysqladmin`:

```
shell> mysqladmin stop-slave
```

Alternatively, you can stop only the slave SQL thread to pause event execution:

```
shell> mysql -e 'STOP SLAVE SQL_THREAD;'
```

This enables the slave to continue to receive data change events from the master's binary log and store them in the relay logs using the I/O thread, but prevents the slave from executing these events and changing its data. Within busy replication environments, permitting the I/O thread to run during backup may speed up the catch-up process when you restart the slave SQL thread.

2. Run `mysqldump` to dump your databases. You may either dump all databases or select databases to be dumped. For example, to dump all databases:

```
shell> mysqldump --all-databases > fulldb.dump
```

3. Once the dump has completed, start slave operations again:

```
shell> mysqladmin start-slave
```

In the preceding example, you may want to add login credentials (user name, password) to the commands, and bundle the process up into a script that you can run automatically each day.

If you use this approach, make sure you monitor the slave replication process to ensure that the time taken to run the backup does not affect the slave's ability to keep up with events from the master. See [Checking Replication Status](#). If the slave is unable to keep up, you may want to add another slave and distribute the backup process. For an example of how to configure this scenario, see [Replicating Different Databases to Different Slaves](#).

## Backing Up Raw Data from a Slave

To guarantee the integrity of the files that are copied, backing up the raw data files on your MySQL replication slave should take place while your slave server is shut down. If the MySQL server is still running, background tasks may still be updating the database files, particularly those involving storage engines with background processes such as `InnoDB`. With `InnoDB`, these problems should be resolved during crash recovery, but since the slave server can be shut down during the backup process without affecting the execution of the master it makes sense to take advantage of this capability.



To shut down the server and back up the files:

1. Shut down the slave MySQL server:

```
shell> mysqladmin shutdown
```

2. Copy the data files. You can use any suitable copying or archive utility, including `cp`, `tar` or `WinZip`. For example, assuming that the data directory is located under the current directory, you can archive the entire directory as follows:

```
shell> tar cf /tmp/dbbackup.tar ./data
```

3. Start the MySQL server again. Under Unix:

```
shell> mysqld_safe &
```

Under Windows:

```
C:\> "C:\Program Files\MySQL\MySQL Server 5.0\bin\mysqld"
```

Normally you should back up the entire data directory for the slave MySQL server. If you want to be able to restore the data and operate as a slave (for example, in the event of failure of the slave), then in addition to the slave's data, you should also back up the slave status files, `master.info` and `relay-log.info`, along with the relay log files. These files are needed to resume replication after you restore the slave's data.

If you lose the relay logs but still have the `relay-log.info` file, you can check it to determine how far the SQL thread has executed in the master binary logs. Then you can use `CHANGE MASTER TO` with the `MASTER_LOG_FILE` and `MASTER_LOG_POS` options to tell the slave to re-read the binary logs from that point. This requires that the binary logs still exist on the master server.

If your slave is replicating `LOAD DATA INFILE` statements, you should also back up any `SQL_LOAD-*` files that exist in the directory that the slave uses for this purpose. The slave needs these files to resume replication of any interrupted `LOAD DATA INFILE` operations. The location of this directory is the value of the `--slave-load-tmpdir` [33] option. If the server was not started with that option, the directory location is the value of the `tmpdir` system variable.

## Using Replication with Different Master and Slave Storage Engines

It does not matter for the replication process whether the source table on the master and the replicated table on the slave use different engine types. In fact, the system variables `storage_engine` and `table_type` are not replicated.

This provides a number of benefits in the replication process in that you can take advantage of different engine types for different replication scenarios. For example, in a typical scale-out scenario (see [Using Replication for Scale-Out](#)), you want to use `InnoDB` tables on the master to take advantage of the transactional functionality, but use `MyISAM` on the slaves where transaction support is not required because the data is only read. When using replication in a data-logging environment you may want to use the `Archive` storage engine on the slave.

Configuring different engines on the master and slave depends on how you set up the initial replication process:

- If you used `mysqldump` to create the database snapshot on your master, you could edit the dump file text to change the engine type used on each table.

Another alternative for `mysqldump` is to disable engine types that you do not want to use on the slave before using the dump to build the data on the slave. For example, you can add the `--skip-`

`innodb` option on your slave to disable the `InnoDB` engine. If a specific engine does not exist for a table to be created, MySQL will use the default engine type, usually `MyISAM`. (This requires that the `NO_ENGINE_SUBSTITUTION` SQL mode is not enabled.) If you want to disable additional engines in this way, you may want to consider building a special binary to be used on the slave that only supports the engines you want.

- If you are using raw data files (a binary backup) to set up the slave, you will be unable to change the initial table format. Instead, use `ALTER TABLE` to change the table types after the slave has been started.
- For new master/slave replication setups where there are currently no tables on the master, avoid specifying the engine type when creating new tables.

If you are already running a replication solution and want to convert your existing tables to another engine type, follow these steps:

1. Stop the slave from running replication updates:

```
mysql> STOP SLAVE;
```

This will enable you to change engine types without interruptions.

2. Execute an `ALTER TABLE ... ENGINE='engine_type'` for each table to be changed.
3. Start the slave replication process again:

```
mysql> START SLAVE;
```

Although the `storage_engine` and `table_type` variables are not replicated, be aware that `CREATE TABLE` and `ALTER TABLE` statements that include the engine specification will be correctly replicated to the slave. For example, if you have a CSV table and you execute:

```
mysql> ALTER TABLE csvtable Engine='MyISAM';
```

The above statement will be replicated to the slave and the engine type on the slave will be converted to `MyISAM`, even if you have previously changed the table type on the slave to an engine other than CSV. If you want to retain engine differences on the master and slave, you should be careful to use the `storage_engine` variable on the master when creating a new table. For example, instead of:

```
mysql> CREATE TABLE tablea (columna int) Engine=MyISAM;
```

Use this format:

```
mysql> SET storage_engine=MyISAM;  
mysql> CREATE TABLE tablea (columna int);
```

When replicated, the `storage_engine` variable will be ignored, and the `CREATE TABLE` statement will execute on the slave using the slave's default engine.

## Using Replication for Scale-Out

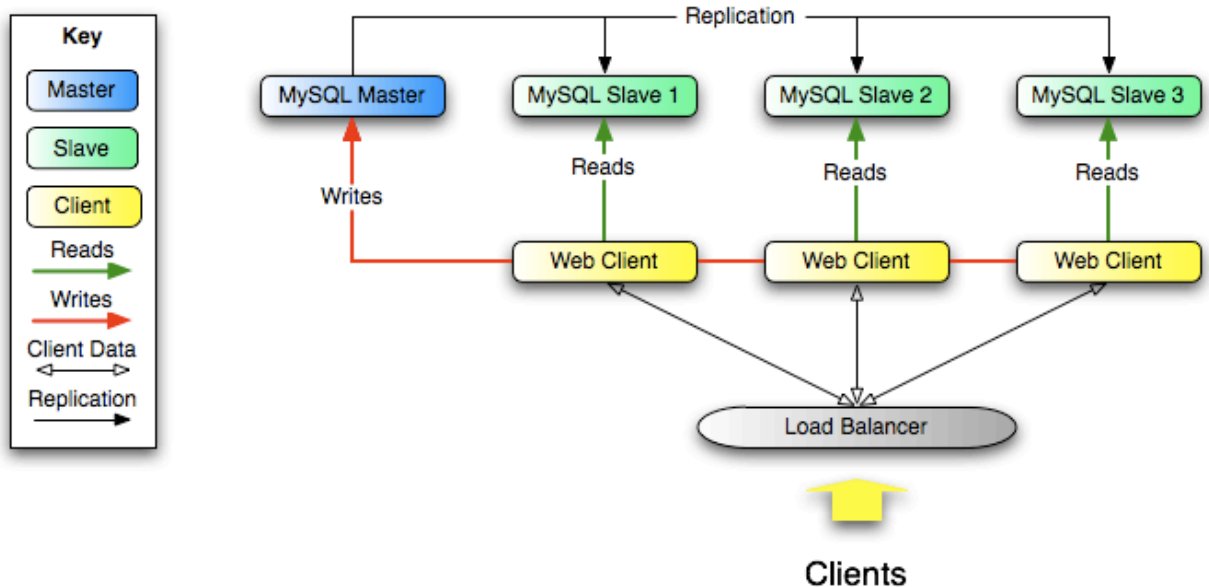
You can use replication as a scale-out solution; that is, where you want to split up the load of database queries across multiple database servers, within some reasonable limitations.

Because replication works from the distribution of one master to one or more slaves, using replication for scale-out works best in an environment where you have a high number of reads and low number of writes/updates. Most Web sites fit into this category, where users are browsing the Web site, reading articles,

posts, or viewing products. Updates only occur during session management, or when making a purchase or adding a comment/message to a forum.

Replication in this situation enables you to distribute the reads over the replication slaves, while still enabling your web servers to communicate with the replication master when a write is required. You can see a sample replication layout for this scenario in [Figure 3.1, “Using Replication to Improve Performance During Scale-Out”](#).

**Figure 3.1. Using Replication to Improve Performance During Scale-Out**



If the part of your code that is responsible for database access has been properly abstracted/modularized, converting it to run with a replicated setup should be very smooth and easy. Change the implementation of your database access to send all writes to the master, and to send reads to either the master or a slave. If your code does not have this level of abstraction, setting up a replicated system gives you the opportunity and motivation to clean it up. Start by creating a wrapper library or module that implements the following functions:

- `safe_writer_connect()`
- `safe_reader_connect()`
- `safe_reader_statement()`
- `safe_writer_statement()`

`safe_` in each function name means that the function takes care of handling all error conditions. You can use different names for the functions. The important thing is to have a unified interface for connecting for reads, connecting for writes, doing a read, and doing a write.

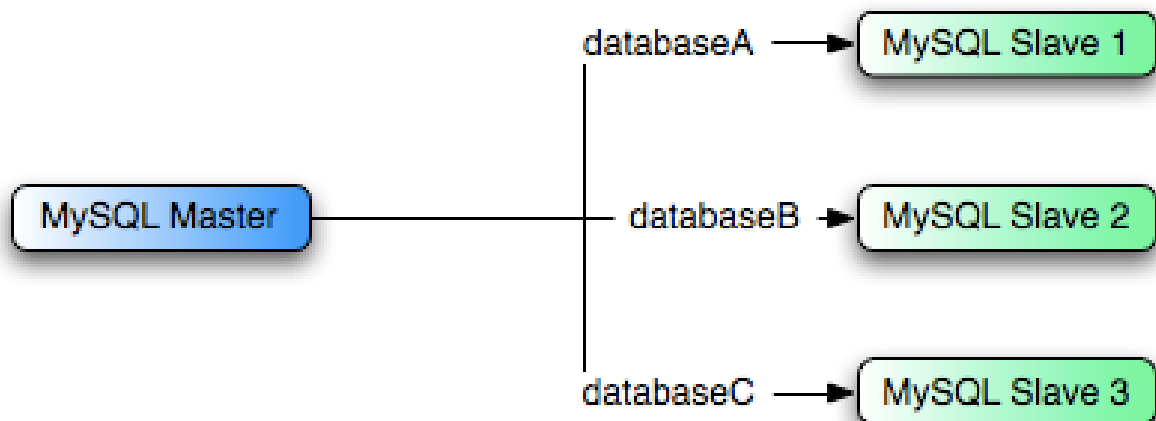
Then convert your client code to use the wrapper library. This may be a painful and scary process at first, but it pays off in the long run. All applications that use the approach just described are able to take advantage of a master/slave configuration, even one involving multiple slaves. The code is much easier to maintain, and adding troubleshooting options is trivial. You need modify only one or two functions; for example, to log how long each statement took, or which statement among those issued gave you an error.

If you have written a lot of code, you may want to automate the conversion task by using the `replace` utility that comes with standard MySQL distributions, or write your own conversion script. Ideally, your code uses consistent programming style conventions. If not, then you are probably better off rewriting it anyway, or at least going through and manually regularizing it to use a consistent style.

## Replicating Different Databases to Different Slaves

There may be situations where you have a single master and want to replicate different databases to different slaves. For example, you may want to distribute different sales data to different departments to help spread the load during data analysis. A sample of this layout is shown in [Figure 3.2, “Using Replication to Replicate Databases to Separate Replication Slaves”](#).

**Figure 3.2. Using Replication to Replicate Databases to Separate Replication Slaves**



You can achieve this separation by configuring the master and slaves as normal, and then limiting the binary log statements that each slave processes by using the `--replicate-wild-do-table` [\[30\]](#) configuration option on each slave.

### Important

You should *not* use `--replicate-do-db` [\[28\]](#) for this purpose, since its affects vary according to the database that is currently selected.

For example, to support the separation as shown in [Figure 3.2, “Using Replication to Replicate Databases to Separate Replication Slaves”](#), you should configure each replication slave as follows, before executing `START SLAVE`:

- Replication slave 1 should use `--replicate-wild-do-table=databaseA.%`.
- Replication slave 2 should use `--replicate-wild-do-table=databaseB.%`.
- Replication slave 3 should use `--replicate-wild-do-table=databaseC.%`.

Each slave in this configuration receives the entire binary log from the master, but executes only those events from the binary log that apply to the databases and tables included by the `--replicate-wild-do-table` [\[30\]](#) option in effect on that slave.

If you have data that must be synchronized to the slaves before replication starts, you have a number of choices:

- Synchronize all the data to each slave, and delete the databases, tables, or both that you do not want to keep.
- Use `mysqldump` to create a separate dump file for each database and load the appropriate dump file on each slave.
- Use a raw data file dump and include only the specific files and databases that you need for each slave.

**Note**

This does not work with `InnoDB` databases unless you use `innodb_file_per_table`.

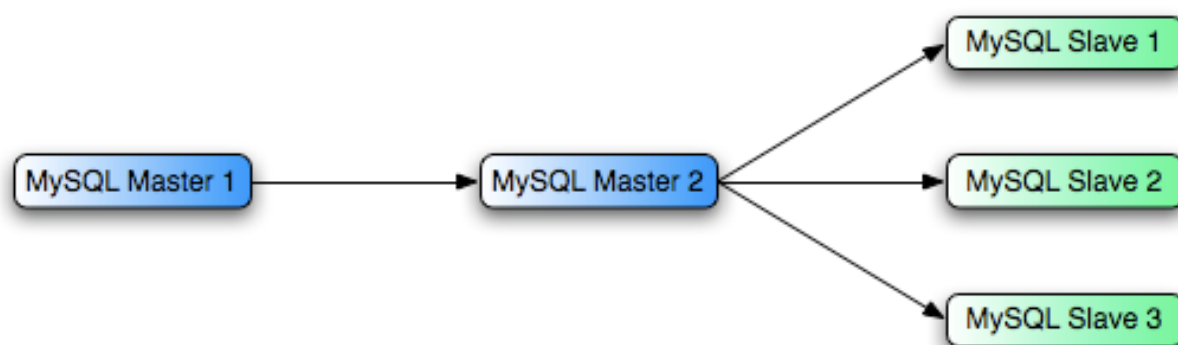
## Improving Replication Performance

As the number of slaves connecting to a master increases, the load, although minimal, also increases, as each slave uses a client connection to the master. Also, as each slave must receive a full copy of the master binary log, the network load on the master may also increase and create a bottleneck.

If you are using a large number of slaves connected to one master, and that master is also busy processing requests (for example, as part of a scale-out solution), then you may want to improve the performance of the replication process.

One way to improve the performance of the replication process is to create a deeper replication structure that enables the master to replicate to only one slave, and for the remaining slaves to connect to this primary slave for their individual replication requirements. A sample of this structure is shown in [Figure 3.3, “Using an Additional Replication Host to Improve Performance”](#).

**Figure 3.3. Using an Additional Replication Host to Improve Performance**



For this to work, you must configure the MySQL instances as follows:

- Master 1 is the primary master where all changes and updates are written to the database. Binary logging should be enabled on this machine.
- Master 2 is the slave to the Master 1 that provides the replication functionality to the remainder of the slaves in the replication structure. Master 2 is the only machine permitted to connect to Master 1. Master

2 also has binary logging enabled, and the `--log-slave-updates` [22] option so that replication instructions from Master 1 are also written to Master 2's binary log so that they can then be replicated to the true slaves.

- Slave 1, Slave 2, and Slave 3 act as slaves to Master 2, and replicate the information from Master 2, which actually consists of the upgrades logged on Master 1.

The above solution reduces the client load and the network interface load on the primary master, which should improve the overall performance of the primary master when used as a direct database solution.

If your slaves are having trouble keeping up with the replication process on the master, there are a number of options available:

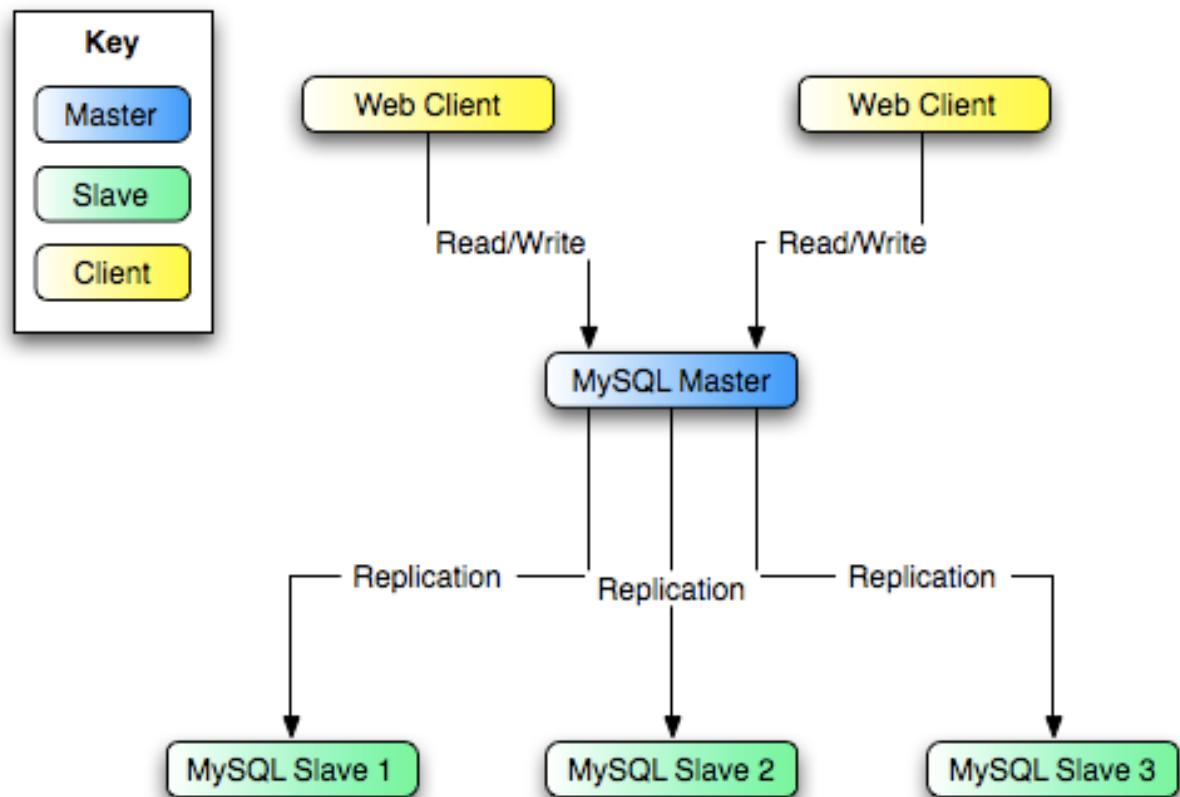
- If possible, put the relay logs and the data files on different physical drives. To do this, use the `--relay-log` [26] option to specify the location of the relay log.
- If the slaves are significantly slower than the master, you may want to divide up the responsibility for replicating different databases to different slaves. See [Replicating Different Databases to Different Slaves](#).
- If your master makes use of transactions and you are not concerned about transaction support on your slaves, use `MyISAM` or another nontransactional engine on the slaves. See [Using Replication with Different Master and Slave Storage Engines](#).
- If your slaves are not acting as masters, and you have a potential solution in place to ensure that you can bring up a master in the event of failure, then you can switch off `--log-slave-updates` [22]. This prevents “dumb” slaves from also logging events they have executed into their own binary log.

## Switching Masters During Failover

There is currently no official solution for providing failover between master and slaves in the event of a failure. With the currently available features, you would have to set up a master and a slave (or several slaves), and to write a script that monitors the master to check whether it is up. Then instruct your applications and the slaves to change master in case of failure.

Remember that you can tell a slave to change its master at any time, using the `CHANGE MASTER TO` statement. The slave will not check whether the databases on the master are compatible with the slave, it will just start reading and executing events from the specified binary log coordinates on the new master. In a failover situation, all the servers in the group are typically executing the same events from the same binary log file, so changing the source of the events should not affect the database structure or integrity providing you are careful.

Run your slaves with the `--log-bin` [39] option and without `--log-slave-updates` [22]. In this way, the slave is ready to become a master as soon as you issue `STOP SLAVE; RESET MASTER`, and `CHANGE MASTER TO` statement on the other slaves. For example, assume that you have the structure shown in [Figure 3.4, “Redundancy Using Replication, Initial Structure”](#).

**Figure 3.4. Redundancy Using Replication, Initial Structure**

In this diagram, the `MySQL Master` holds the master database, the `MySQL Slave` hosts are replication slaves, and the `Web Client` machines are issuing database reads and writes. Web clients that issue only reads (and would normally be connected to the slaves) are not shown, as they do not need to switch to a new server in the event of failure. For a more detailed example of a read/write scale-out replication structure, see [Using Replication for Scale-Out](#).

Each `MySQL Slave` (`Slave 1`, `Slave 2`, and `Slave 3`) is a slave running with `--log-bin` [39] and without `--log-slave-updates` [22]. Because updates received by a slave from the master are not logged in the binary log unless `--log-slave-updates` [22] is specified, the binary log on each slave is empty initially. If for some reason `MySQL Master` becomes unavailable, you can pick one of the slaves to become the new master. For example, if you pick `Slave 1`, all `Web Clients` should be redirected to `Slave 1`, which will log updates to its binary log. `Slave 2` and `Slave 3` should then replicate from `Slave 1`.

The reason for running the slave without `--log-slave-updates` [22] is to prevent slaves from receiving updates twice in case you cause one of the slaves to become the new master. Suppose that `Slave 1` has `--log-slave-updates` [22] enabled. Then it will write updates that it receives from `Master` to its own binary log. When `Slave 2` changes from `Master` to `Slave 1` as its master, it may receive updates from `Slave 1` that it has already received from `Master`.

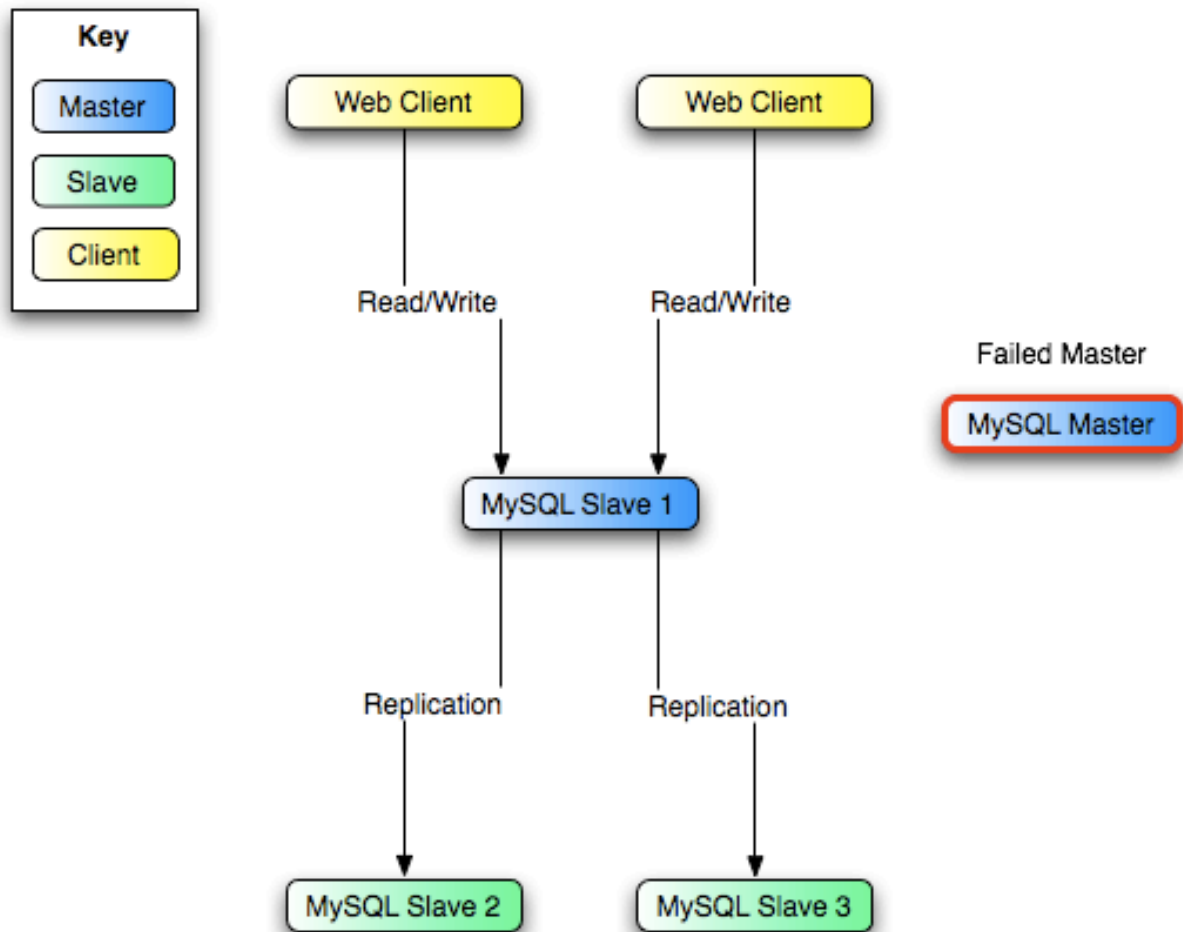
Make sure that all slaves have processed any statements in their relay log. On each slave, issue `STOP SLAVE IO_THREAD`, then check the output of `SHOW PROCESSLIST` until you see `Has read all relay log`. When this is true for all slaves, they can be reconfigured to the new setup. On the slave `Slave 1` being promoted to become the master, issue `STOP SLAVE` and `RESET MASTER`.

On the other slaves `Slave 2` and `Slave 3`, use `STOP SLAVE` and `CHANGE MASTER TO MASTER_HOST='Slave1'` (where `'Slave1'` represents the real host name of `Slave 1`). To use `CHANGE MASTER TO`, add all information about how to connect to `Slave 1` from `Slave 2` or `Slave 3` (`user`, `password`, `port`). In `CHANGE MASTER TO`, there is no need to specify the name of the `Slave 1` binary log file or log position to read from: We know it is the first binary log file and position 4, which are the defaults for `CHANGE MASTER TO`. Finally, use `START SLAVE` on `Slave 2` and `Slave 3`.

Once the new replication is in place, you will then need to instruct each `Web Client` to direct its statements to `Slave 1`. From that point on, all updates statements sent by `Web Client` to `Slave 1` are written to the binary log of `Slave 1`, which then contains every update statement sent to `Slave 1` since `Master` died.

The resulting server structure is shown in [Figure 3.5, “Redundancy Using Replication, After Master Failure”](#).

**Figure 3.5. Redundancy Using Replication, After Master Failure**



When `Master` is up again, you must issue on it the same `CHANGE MASTER TO` as that issued on `Slave 2` and `Slave 3`, so that `Master` becomes a slave of `S1` and picks up each `Web Client` writes that it missed while it was down.

To make `Master` a master again (for example, because it is the most powerful machine), use the preceding procedure as if `Slave 1` was unavailable and `Master` was to be the new master. During this



procedure, do not forget to run `RESET MASTER` on `Master` before making `Slave 1`, `Slave 2`, and `Slave 3` slaves of `Master`. Otherwise, they may pick up old `Web Client` writes from before the point at which `Master` became unavailable.

Note that there is no synchronization between the different slaves to a master. Some slaves might be ahead of others. This means that the concept outlined in the previous example might not work. In practice, however, the relay logs of different slaves will most likely not be far behind the master, so it would work, anyway (but there is no guarantee).

A good way to keep your applications informed as to the location of the master is by having a dynamic DNS entry for the master. With `bind` you can use `nsupdate` to dynamically update your DNS.

## Setting Up Replication Using SSL

To use SSL for encrypting the transfer of the binary log required during replication, both the master and the slave must support SSL network connections. If either host does not support SSL connections (because it has not been compiled or configured for SSL), replication through an SSL connection is not possible.

Setting up replication using an SSL connection is similar to setting up a server and client using SSL. You must obtain (or create) a suitable security certificate that you can use on the master, and a similar certificate (from the same certificate authority) on each slave.

For more information on setting up a server and client for SSL connectivity, see [Configuring MySQL for SSL](#).

To enable SSL on the master you must create or obtain suitable certificates, and then add the following configuration options to the master's configuration within the `[mysqld]` section of the master's `my.cnf` file:

```
[mysqld]
ssl-ca=cacert.pem
ssl-cert=server-cert.pem
ssl-key=server-key.pem
```

The paths to the certificates may be relative or absolute; we recommend that you always use complete paths for this purpose.

The options are as follows:

- `ssl-ca` identifies the Certificate Authority (CA) certificate.
- `ssl-cert` identifies the server public key. This can be sent to the client and authenticated against the CA certificate that it has.
- `ssl-key` identifies the server private key.

On the slave, you have two options available for setting the SSL information. You can either add the slave certificates to the `[client]` section of the slave's `my.cnf` file, or you can explicitly specify the SSL information using the `CHANGE MASTER TO` statement:

- To add the slave certificates using an option file, add the following lines to the `[client]` section of the slave's `my.cnf` file:

```
[client]
ssl-ca=cacert.pem
ssl-cert=client-cert.pem
ssl-key=client-key.pem
```

Restart the slave server, using the `--skip-slave-start` [33] option to prevent the slave from connecting to the master. Use `CHANGE MASTER TO` to specify the master configuration, using the `MASTER_SSL` option to enable SSL connectivity:

```
mysql> CHANGE MASTER TO
-> MASTER_HOST='master_hostname',
-> MASTER_USER='replicate',
-> MASTER_PASSWORD='password',
-> MASTER_SSL=1;
```

- To specify the SSL certificate options using the `CHANGE MASTER TO` statement, append the SSL options:

```
mysql> CHANGE MASTER TO
-> MASTER_HOST='master_hostname',
-> MASTER_USER='replicate',
-> MASTER_PASSWORD='password',
-> MASTER_SSL=1,
-> MASTER_SSL_CA = 'ca_file_name',
-> MASTER_SSL_CAPATH = 'ca_directory_name',
-> MASTER_SSL_CERT = 'cert_file_name',
-> MASTER_SSL_KEY = 'key_file_name';
```

After the master information has been updated, start the slave replication process:

```
mysql> START SLAVE;
```

You can use the `SHOW SLAVE STATUS` statement to confirm that the SSL connection was established successfully.

For more information on the `CHANGE MASTER TO` statement, see [CHANGE MASTER TO Syntax](#).

If you want to enforce the use of SSL connections during replication, then create a user with the `REPLICATION SLAVE` privilege and use the `REQUIRE SSL` option for that user. For example:

```
mysql> CREATE USER 'repl'@'%.mydomain.com' IDENTIFIED BY 'slavepass';
mysql> GRANT REPLICATION SLAVE ON *.*
-> TO 'repl'@'%.mydomain.com' REQUIRE SSL;
```

If the account already exists, you can add `REQUIRE SSL` to it with this statement:

```
mysql> GRANT USAGE ON *.*
-> TO 'repl'@'%.mydomain.com' REQUIRE SSL;
```

---

## Chapter 4. Replication Notes and Tips

### Table of Contents

Replication Features and Issues .....	61
Replication and <code>AUTO_INCREMENT</code> .....	62
Replication and Character Sets .....	63
Replication of <code>CREATE TABLE ... SELECT</code> Statements .....	63
Replication of <code>DROP ... IF EXISTS</code> Statements .....	63
Replication and <code>DIRECTORY</code> Table Options .....	64
Replication and Floating-Point Values .....	64
Replication and <code>FLUSH</code> .....	64
Replication and System Functions .....	64
Replication and <code>LIMIT</code> .....	66
Replication and <code>LOAD</code> Operations .....	66
Replication and the Slow Query Log .....	66
Replication and <code>REPAIR TABLE</code> .....	66
Replication and Master or Slave Shutdowns .....	67
Replication and <code>MEMORY</code> Tables .....	67
Replication and Temporary Tables .....	68
Replication of the <code>mysql</code> System Database .....	68
Replication and User Privileges .....	69
Replication and the Query Optimizer .....	69
Replication and Reserved Words .....	69
Slave Errors During Replication .....	69
Replication and Server SQL Mode .....	70
Replication Retries and Timeouts .....	70
Replication and <code>TIMESTAMP</code> .....	70
Replication and Time Zones .....	70
Replication and Transactions .....	71
Replication and Triggers .....	71
Replication and Views .....	72
Replication and Variables .....	72
Replication Compatibility Between MySQL Versions .....	73
Upgrading a Replication Setup .....	74
Troubleshooting Replication .....	75
How to Report Replication Bugs or Problems .....	76

### Replication Features and Issues

The following sections provide information about what is supported and what is not in MySQL replication, and about specific issues and situations that may occur when replicating certain statements.

Statement-based replication depends on compatibility at the SQL level between the master and slave. In others, successful SBR requires that any SQL features used be supported by both the master and the slave servers. For example, if you use a feature on the master server that is available only in MySQL 5.0 (or later), you cannot replicate to a slave that uses MySQL 4.1 (or earlier).

Such incompatibilities also can occur within a release series when using pre-production releases of MySQL. For example, the `SLEEP()` function is available beginning with MySQL 5.0.12. If you use this function on the master, you cannot replicate to a slave that uses MySQL 5.0.11 or earlier.

For this reason, use Generally Available (GA) releases of MySQL for statement-based replication in a production setting, since we do not introduce new SQL statements or change their behavior within a given release series once that series reaches GA release status.

If you are planning to use replication between MySQL 5.0 and a previous MySQL release series, it is also a good idea to consult the edition of the *MySQL Reference Manual* corresponding to the earlier release series for information regarding the replication characteristics of that series.

For additional information specific to replication and `InnoDB`, see [InnoDB and MySQL Replication](#).

## Replication and `AUTO_INCREMENT`

Replication of `AUTO_INCREMENT`, `LAST_INSERT_ID()`, and `TIMESTAMP` values is done correctly, subject to the following exceptions.

- `INSERT DELAYED ... VALUES(LAST_INSERT_ID())` inserts a different value on the master and the slave. (Bug #20819) This is fixed in MySQL 5.1 when using row-based or mixed-format binary logging. For more information, see [Replication Formats](#).
- Before MySQL 5.0.26, a stored procedure that uses `LAST_INSERT_ID()` does not replicate properly.
- When a statement uses a stored function that inserts into an `AUTO_INCREMENT` column, the generated `AUTO_INCREMENT` value is not written into the binary log, so a different value can in some cases be inserted on the slave. This is also true of a trigger that causes an `INSERT` into an `AUTO_INCREMENT` column.
- An insert into an `AUTO_INCREMENT` column caused by a stored routine or trigger running on a master that uses MySQL 5.0.60 or earlier does not replicate correctly to a slave running MySQL 5.1.12 through 5.1.23 (inclusive). (Bug #33029)
- An `INSERT` into a table that has a composite primary key that includes an `AUTO_INCREMENT` column that is not the first column of this composite key is not logged or replicated correctly.

This issue does not affect tables using the `InnoDB` storage engine, since `InnoDB` does not allow the creation of a composite key that includes an `AUTO_INCREMENT` column that is not the first column in the key.

- Adding an `AUTO_INCREMENT` column to a table with `ALTER TABLE` might not produce the same ordering of the rows on the slave and the master. This occurs because the order in which the rows are numbered depends on the specific storage engine used for the table and the order in which the rows were inserted. If it is important to have the same order on the master and slave, the rows must be ordered before assigning an `AUTO_INCREMENT` number. Assuming that you want to add an `AUTO_INCREMENT` column to a table `t1` that has columns `col1` and `col2`, the following statements produce a new table `t2` identical to `t1` but with an `AUTO_INCREMENT` column:

```
CREATE TABLE t2 LIKE t1;
ALTER TABLE t2 ADD id INT AUTO_INCREMENT PRIMARY KEY;
INSERT INTO t2 SELECT * FROM t1 ORDER BY col1, col2;
```

### Important

To guarantee the same ordering on both master and slave, the `ORDER BY` clause must name *all* columns of `t1`.

The instructions just given are subject to the limitations of `CREATE TABLE ... LIKE`: Foreign key definitions are ignored, as are the `DATA DIRECTORY` and `INDEX DIRECTORY` table options. If a table definition includes any of those characteristics, create `t2` using a `CREATE TABLE` statement that is identical to the one used to create `t1`, but with the addition of the `AUTO_INCREMENT` column.

Regardless of the method used to create and populate the copy having the `AUTO_INCREMENT` column, the final step is to drop the original table and then rename the copy:

```
DROP t1;  
ALTER TABLE t2 RENAME t1;
```

See also [Problems with ALTER TABLE](#).

## Replication and Character Sets

The following applies to replication between MySQL servers that use different character sets:

- If the master uses MySQL 4.1, you must *always* use the same *global* character set and collation on the master and the slave, regardless of the slave MySQL version. (These are controlled by the `--character-set-server` and `--collation-server` options.) Otherwise, you may get duplicate-key errors on the slave, because a key that is unique in the master character set might not be unique in the slave character set. Note that this is not a cause for concern when master and slave are both MySQL 5.0 or later.
- If the master is older than MySQL 4.1.3, the character set of any client should never be made different from its global value because this character set change is not known to the slave. In other words, clients should not use `SET NAMES`, `SET CHARACTER SET`, and so forth. If both the master and the slave are 4.1.3 or newer, clients can freely set session values for character set variables because these settings are written to the binary log and so are known to the slave. That is, clients can use `SET NAMES` or `SET CHARACTER SET` or can set variables such as `collation_client` or `collation_server`. However, clients are prevented from changing the *global* value of these variables; as stated previously, the master and slave must always have identical global character set values.
- If the master has databases with a character set different from the global `character_set_server` value, you should design your `CREATE TABLE` statements so that they do not implicitly rely on the database default character set. A good workaround is to state the character set and collation explicitly in `CREATE TABLE` statements.

## Replication of `CREATE TABLE ... SELECT` Statements

This section discusses the rules that are applied when a `CREATE TABLE ... SELECT` statement is replicated.

### Note

`CREATE TABLE ... SELECT` always performs an implicit commit ([Statements That Cause an Implicit Commit](#)).

**Statement succeeds.** A successful `CREATE TABLE ... SELECT` is itself replicated.

**Statement fails.** A failed `CREATE TABLE ... SELECT` replicates as follows:

- **Statement does not use `IF NOT EXISTS`.** The statement has no effect. However, the implicit commit caused by the statement is logged. This is true regardless of the storage engine used and the reason for which the statement failed.
- **Statement uses `IF NOT EXISTS`.** The `CREATE TABLE IF NOT EXISTS ... SELECT` is logged with an error.

## Replication of `DROP ... IF EXISTS` Statements

The `DROP DATABASE IF EXISTS`, `DROP TABLE IF EXISTS`, and `DROP VIEW IF EXISTS` statements are always replicated, even if the database, table, or view to be dropped does not exist on the master. This is to ensure that the object to be dropped no longer exists on either the master or the slave, once the slave has caught up with the master.

Beginning with MySQL 5.0.82, `DROP ... IF EXISTS` statements for stored programs (stored procedures and functions, triggers, and events) are also replicated, even if the stored program to be dropped does not exist on the master. (Bug #13684)

## Replication and `DIRECTORY` Table Options

If a `DATA DIRECTORY` or `INDEX DIRECTORY` table option is used in a `CREATE TABLE` statement on the master server, the table option is also used on the slave. This can cause problems if no corresponding directory exists in the slave host file system or if it exists but is not accessible to the slave server. This can be overridden by using the `NO_DIR_IN_CREATE` server SQL mode on the slave, which causes the slave to ignore the `DATA DIRECTORY` and `INDEX DIRECTORY` table options when replicating `CREATE TABLE` statements. The result is that `MyISAM` data and index files are created in the table's database directory.

For more information, see [Server SQL Modes](#).

## Replication and Floating-Point Values

With statement-based replication, values are converted from decimal to binary. Because conversions between decimal and binary representations of them may be approximate, comparisons involving floating-point values are inexact. This is true for operations that use floating-point values explicitly, or that use values that are converted to floating-point implicitly. Comparisons of floating-point values might yield different results on master and slave servers due to differences in computer architecture, the compiler used to build MySQL, and so forth. See [Type Conversion in Expression Evaluation](#), and [Problems with Floating-Point Values](#).

## Replication and `FLUSH`

Some forms of the `FLUSH` statement are not logged because they could cause problems if replicated to a slave: `FLUSH LOGS`, `FLUSH MASTER`, `FLUSH SLAVE`, and `FLUSH TABLES WITH READ LOCK`. For a syntax example, see [FLUSH Syntax](#). The `FLUSH TABLES`, `ANALYZE TABLE`, `OPTIMIZE TABLE`, and `REPAIR TABLE` statements are written to the binary log and thus replicated to slaves. This is not normally a problem because these statements do not modify table data.

However, this behavior can cause difficulties under certain circumstances. If you replicate the privilege tables in the `mysql` database and update those tables directly without using `GRANT`, you must issue a `FLUSH PRIVILEGES` on the slaves to put the new privileges into effect. In addition, if you use `FLUSH TABLES` when renaming a `MyISAM` table that is part of a `MERGE` table, you must issue `FLUSH TABLES` manually on the slaves. These statements are written to the binary log unless you specify `NO_WRITE_TO_BINLOG` or its alias `LOCAL`.

## Replication and System Functions

Certain functions do not replicate well under some conditions:

- The `USER()`, `CURRENT_USER()`, `UUID()`, `VERSION()`, and `LOAD_FILE()` functions are replicated without change and thus do not work reliably on the slave.
- For `NOW()`, the binary log includes the timestamp. This means that the value *as returned by the call to this function on the master* is replicated to the slave. This can lead to a possibly unexpected result when replicating between MySQL servers in different time zones. Suppose that the master is located in New

York, the slave is located in Stockholm, and both servers are using local time. Suppose further that, on the master, you create a table `mytable`, perform an `INSERT` statement on this table, and then select from the table, as shown here:

```
mysql> CREATE TABLE mytable (mycol TEXT);
Query OK, 0 rows affected (0.06 sec)
mysql> INSERT INTO mytable VALUES ( NOW() );
Query OK, 1 row affected (0.00 sec)
mysql> SELECT * FROM mytable;
+-----+
| mycol                |
+-----+
| 2009-09-01 12:00:00 |
+-----+
1 row in set (0.00 sec)
```

Local time in Stockholm is 6 hours later than in New York; so, if you issue `SELECT NOW()` on the slave at that exact same instant, the value `2009-09-01 18:00:00` is returned. For this reason, if you select from the slave's copy of `mytable` after the `CREATE TABLE` and `INSERT` statements just shown have been replicated, you might expect `mycol` to contain the value `2009-09-01 18:00:00`. However, this is not the case; when you select from the slave's copy of `mytable`, you obtain exactly the same result as on the master:

```
mysql> SELECT * FROM mytable;
+-----+
| mycol                |
+-----+
| 2009-09-01 12:00:00 |
+-----+
1 row in set (0.00 sec)
```

As of MySQL 5.0.13, the `SYSDATE()` function is no longer equivalent to `NOW()`. Implications are that `SYSDATE()` is not replication-safe because it is not affected by `SET TIMESTAMP` statements in the binary log and is nondeterministic. To avoid this, you can start the server with the `--sysdate-is-now` option to cause `SYSDATE()` to be an alias for `NOW()`.

See also [Replication and Time Zones](#).

- The `GET_LOCK()`, `RELEASE_LOCK()`, `IS_FREE_LOCK()`, and `IS_USED_LOCK()` functions that handle user-level locks are replicated without the slave knowing the concurrency context on the master. Therefore, these functions should not be used to insert into a master table because the content on the slave would differ. For example, do not issue a statement such as `INSERT INTO mytable VALUES (GET_LOCK(...))`.

As a workaround for the preceding limitations, you can use the strategy of saving the problematic function result in a user variable and referring to the variable in a later statement. For example, the following single-row `INSERT` is problematic due to the reference to the `UUID()` function:

```
INSERT INTO t VALUES(UUID());
```

To work around the problem, do this instead:

```
SET @my_uuid = UUID();
INSERT INTO t VALUES(@my_uuid);
```

That sequence of statements replicates because the value of `@my_uuid` is stored in the binary log as a user-variable event prior to the `INSERT` statement and is available for use in the `INSERT`.

The same idea applies to multiple-row inserts, but is more cumbersome to use. For a two-row insert, you can do this:



```
SET @my_uuid1 = UUID(); @my_uuid2 = UUID();
INSERT INTO t VALUES(@my_uuid1),(@my_uuid2);
```

However, if the number of rows is large or unknown, the workaround is difficult or impracticable. For example, you cannot convert the following statement to one in which a given individual user variable is associated with each row:

```
INSERT INTO t2 SELECT UUID(), * FROM t1;
```

Non-delayed **INSERT** statements that refer to **RAND()** or user-defined variables replicate correctly. However, changing the statements to use **INSERT DELAYED** can result in different results on master and slave.

Within a stored function, **RAND()** replicates correctly as long as it is invoked only once during the execution of the function. (You can consider the function execution timestamp and random number seed as implicit inputs that are identical on the master and slave.)

The **FOUND\_ROWS()** and **ROW\_COUNT()** functions are also not replicated reliably. A workaround is to store the result of the function call in a user variable, and then use that in the **INSERT** statement. For example, if you wish to store the result in a table named **mytable**, you might normally do so like this:

```
SELECT SQL_CALC_FOUND_ROWS FROM mytable LIMIT 1;
INSERT INTO mytable VALUES( FOUND_ROWS() );
```

However, if you are replicating **mytable**, you should use **SELECT ... INTO**, and then store the variable in the table, like this:

```
SELECT SQL_CALC_FOUND_ROWS INTO @found_rows FROM mytable LIMIT 1;
INSERT INTO mytable VALUES(@found_rows);
```

In this way, the user variable is replicated as part of the context, and applied on the slave correctly.

## Replication and **LIMIT**

Replication of **LIMIT** clauses in **DELETE**, **UPDATE**, and **INSERT ... SELECT** statements is not guaranteed, since the order of the rows affected is not defined. Such statements can be replicated correctly only if they also contain an **ORDER BY** clause.

## Replication and **LOAD** Operations

Using **LOAD TABLE FROM MASTER** where the master is running MySQL 4.1 and the slave is running MySQL 5.0 may corrupt the table data, and is not supported. (Bug #16261)

The **LOAD DATA INFILE** statement **CONCURRENT** option is not replicated; that is, **LOAD DATA CONCURRENT INFILE** is replicated as **LOAD DATA INFILE**, and **LOAD DATA CONCURRENT LOCAL INFILE** is replicated as **LOAD DATA LOCAL INFILE**. (Bug #34628)

*The following applies only if either the master or the slave is running MySQL 5.0.3 or older.* If on the master a **LOAD DATA INFILE** is interrupted (integrity constraint violation, killed connection, and so on), the slave skips the **LOAD DATA INFILE** entirely. This means that if this command permanently inserted or updated table records before being interrupted, these modifications are not replicated to the slave.

## Replication and the Slow Query Log

Replication slaves do not write replicated queries to the slow query log, even if the same queries were written to the slow query log on the master. This is a known issue. (Bug #23300)

## Replication and **REPAIR TABLE**



When used on a corrupted or otherwise damaged table, it is possible for the `REPAIR TABLE` statement to delete rows that cannot be recovered. However, any such modifications of table data performed by this statement are not replicated, which can cause master and slave to lose synchronization. For this reason, in the event that a table on the master becomes damaged and you use `REPAIR TABLE` to repair it, you should first stop replication (if it is still running) before using `REPAIR TABLE`, then afterward compare the master's and slave's copies of the table and be prepared to correct any discrepancies manually, before restarting replication.

## Replication and Master or Slave Shutdowns

It is safe to shut down a master server and restart it later. When a slave loses its connection to the master, the slave tries to reconnect immediately and retries periodically if that fails. The default is to retry every 60 seconds. This may be changed with the `CHANGE MASTER TO` statement or `--master-connect-retry` [23] option. A slave also is able to deal with network connectivity outages. However, the slave notices the network outage only after receiving no data from the master for `slave_net_timeout` [37] seconds. If your outages are short, you may want to decrease `slave_net_timeout` [37]. See [Server System Variables](#).

An unclean shutdown (for example, a crash) on the master side can result in the master binary log having a final position less than the most recent position read by the slave, due to the master binary log file not being flushed. This can cause the slave not to be able to replicate when the master comes back up. Setting `sync_binlog=1` [44] in the master `my.cnf` file helps to minimize this problem because it causes the master to flush its binary log more frequently.

Unclean master shutdowns may cause inconsistencies between the content of tables and the binary log. This can be avoided by using `InnoDB` tables and the `--innodb-safe-binlog` option on the master. See [The Binary Log](#).

### Note

`--innodb-safe-binlog` is unneeded as of MySQL 5.0.3, having been made obsolete by the introduction of XA transaction support.

Shutting down a slave cleanly is safe because it keeps track of where it left off. However, be careful that the slave does not have temporary tables open; see [Replication and Temporary Tables](#). Unclean shutdowns might produce problems, especially if the disk cache was not flushed to disk before the problem occurred:

- For transactions, the slave commits and then updates `relay-log.info`. If a crash occurs between these two operations, relay log processing will have proceeded further than the information file indicates and the slave will re-execute the events from the last transaction in the relay log after it has been restarted.
- A similar problem can occur if the slave updates `relay-log.info` but the server host crashes before the write has been flushed to disk. Writes are not forced to disk because the server relies on the operating system to flush the file from time to time.

The fault tolerance of your system for these types of problems is greatly increased if you have a good uninterruptible power supply.

## Replication and `MEMORY` Tables

When a master server shuts down and restarts, its `MEMORY` (`HEAP`) tables become empty. To replicate this effect to slaves, the first time that the master uses a given `MEMORY` table after startup, it logs an event that notifies slaves that the table must to be emptied by writing a `DELETE` statement for that table to the binary log.

When a slave server shuts down and restarts, its `MEMORY` tables become empty. This causes the slave to be out of synchrony with the master and may lead to other failures or cause the slave to stop. For example, `INSERT INTO ... SELECT FROM memory_table` may insert a different set of rows on the master and slave.

The safe way to restart a slave that is replicating `MEMORY` tables is to first drop or delete all rows from the `MEMORY` tables on the master and wait until those changes have replicated to the slave. Then it is safe to restart the slave.

The size of `MEMORY` tables is limited by the value of the `max_heap_table_size` system variable, which is not replicated (see [Replication and Variables](#)). A change in `max_heap_table_size` takes effect for `MEMORY` tables that are created or updated using `ALTER TABLE ... ENGINE = MEMORY` or `TRUNCATE TABLE` following the change, or for all `MEMORY` tables following a server restart. If you increase the value of this variable on the master without doing so on the slave, it becomes possible for a table on the master to grow larger than its counterpart on the slave, leading to inserts that succeed on the master but fail on the slave with `Table is full` errors. This is a known issue (Bug #48666). In such cases, you must set the global value of `max_heap_table_size` on the slave as well as on the master, then restart replication. It is also recommended that you restart both the master and slave MySQL servers, to insure that the new value takes complete (global) effect on each of them.

See [The MEMORY \(HEAP\) Storage Engine](#), for more information about `MEMORY` tables.

## Replication and Temporary Tables

**Safe slave shutdown when using temporary tables.** Temporary tables are replicated except in the case where you stop the slave server (not just the slave threads) and you have replicated temporary tables that are open for use in updates that have not yet been executed on the slave. If you stop the slave server, the temporary tables needed by those updates are no longer available when the slave is restarted. To avoid this problem, do not shut down the slave while it has temporary tables open. Instead, use the following procedure:

1. Issue a `STOP SLAVE SQL_THREAD` statement.
2. Use `SHOW STATUS` to check the value of the `Slave_open_temp_tables` variable.
3. If the value is not 0, restart the slave SQL thread with `START SLAVE SQL_THREAD` and repeat the procedure later.
4. When the value is 0, issue a `mysqladmin shutdown` command to stop the slave.

**Temporary tables and replication options.** By default, all temporary tables are replicated; this happens whether or not there are any matching `--replicate-do-db` [28], `--replicate-do-table` [29], or `--replicate-wild-do-table` [30] options in effect. However, the `--replicate-ignore-table` [29] and `--replicate-wild-ignore-table` [31] options are honored for temporary tables.

A recommended practice when using replication is to designate a prefix for exclusive use in naming temporary tables that you do not want replicated, then employ a matching `--replicate-wild-ignore-table` [31] option. For example, you might give all such tables names beginning with `norep` (such as `norepmytable`, `norepyourtable`, and so on), then use `--replicate-wild-ignore-table=norep%` [31] to prevent the replication of these tables.

## Replication of the `mysql` System Database

User privileges are replicated only if the `mysql` database is replicated. That is, the `GRANT`, `REVOKE`, `SET PASSWORD`, `CREATE USER`, and `DROP USER` statements take effect on the slave only if the replication setup includes the `mysql` database.

## Replication and User Privileges

User privileges are replicated only if the `mysql` database is replicated. That is, the `GRANT`, `REVOKE`, `SET PASSWORD`, `CREATE USER`, and `DROP USER` statements take effect on the slave only if the replication setup includes the `mysql` database.

If you are replicating all databases, but do not want statements that affect user privileges to be replicated, set up the slave not to replicate the `mysql` database, using the `--replicate-wild-ignore-table=mysql.%` [31] option. The slave recognizes that privilege-related SQL statements have no effect, and thus it does not execute those statements.

## Replication and the Query Optimizer

It is possible for the data on the master and slave to become different if a statement is written in such a way that the data modification is nondeterministic; that is, left up to the query optimizer. (In general, this is not a good practice, even outside of replication.) Examples of nondeterministic statements include `DELETE` or `UPDATE` statements that use `LIMIT` with no `ORDER BY` clause; see [Replication and LIMIT](#), for a detailed discussion of these.

Also see [Known Issues in MySQL](#).

## Replication and Reserved Words

You can encounter problems when you attempt to replicate from an older master to a newer slave and you make use of identifiers on the master that are reserved words in the newer MySQL version running on the slave. An example of this is using a table column named `current_user` on a 4.0 master that is replicating to a 4.1 or higher slave because `CURRENT_USER` is a reserved word beginning in MySQL 4.1. Replication can fail in such cases with Error 1064 *You have an error in your SQL syntax..., even if a database or table named using the reserved word or a table having a column named using the reserved word is excluded from replication*. This is due to the fact that each SQL event must be parsed by the slave prior to execution, so that the slave knows which database object or objects would be affected; only after the event is parsed can the slave apply any filtering rules defined by `--replicate-do-db` [28], `--replicate-do-table` [29], `--replicate-ignore-db` [28], and `--replicate-ignore-table` [29].

To work around the problem of database, table, or column names on the master which would be regarded as reserved words by the slave, do one of the following:

- Use one or more `ALTER TABLE` statements on the master to change the names of any database objects where these names would be considered reserved words on the slave, and change any SQL statements that use the old names to use the new names instead.
- In any SQL statements using these database object names, write the names as quoted identifiers using backtick characters (```).

For listings of reserved words by MySQL version, see [Reserved Words](#), in the *MySQL Server Version Reference*. For identifier quoting rules, see [Schema Object Names](#).

## Slave Errors During Replication

If a statement produces the same error (identical error code) on both the master and the slave, the error is logged, but replication continues.

If a statement produces different errors on the master and the slave, the slave SQL thread terminates, and the slave writes a message to its error log and waits for the database administrator to decide what to do about the error. This includes the case that a statement produces an error on the master or the slave, but

not both. To address the issue, connect to the slave manually and determine the cause of the problem. `SHOW SLAVE STATUS` is useful for this. Then fix the problem and run `START SLAVE`. For example, you might need to create a nonexistent table before you can start the slave again.

If this error code validation behavior is not desirable, some or all errors can be masked out (ignored) with the `--slave-skip-errors` [34] option.

For nontransactional storage engines such as `MyISAM`, it is possible to have a statement that only partially updates a table and returns an error code. This can happen, for example, on a multiple-row insert that has one row violating a key constraint, or if a long update statement is killed after updating some of the rows. If that happens on the master, the slave expects execution of the statement to result in the same error code. If it does not, the slave SQL thread stops as described previously.

If you are replicating between tables that use different storage engines on the master and slave, keep in mind that the same statement might produce a different error when run against one version of the table, but not the other, or might cause an error for one version of the table, but not the other. For example, since `MyISAM` ignores foreign key constraints, an `INSERT` or `UPDATE` statement accessing an `InnoDB` table on the master might cause a foreign key violation but the same statement performed on a `MyISAM` version of the same table on the slave would produce no such error, causing replication to stop.

## Replication and Server SQL Mode

Using different server SQL mode settings on the master and the slave may cause the same `INSERT` statements to be handled differently on the master and the slave, leading the master and slave to diverge. For best results, you should always use the same server SQL mode on the master and on the slave.

For more information, see [Server SQL Modes](#).

## Replication Retries and Timeouts

In MySQL 5.0 (starting from 5.0.3), there is a global system variable `slave_transaction_retries` [38]: If the slave SQL thread fails to execute a transaction because of an `InnoDB` deadlock or because it exceeded the `InnoDB innodb_lock_wait_timeout` or the `NDBCLUSTER TransactionDeadlockDetectionTimeout` or `TransactionInactiveTimeout` value, the slave automatically retries the transaction `slave_transaction_retries` [38] times before stopping with an error. The default value is 10. Starting from MySQL 5.0.4, the total retry count can be seen in the output of `SHOW STATUS`; see [Server Status Variables](#).

## Replication and `TIMESTAMP`

Older versions of MySQL (prior to 4.1) differed significantly in several ways in their handling of the `TIMESTAMP` data type from what is supported in MySQL versions 5.0 and newer; these include syntax extensions which are deprecated in MySQL 5.1, and that no longer supported in MySQL 5.5. This this can cause problems (including replication failures) when replicating between MySQL Server versions, if you are using columns that are defined using the old `TIMESTAMP(N)` syntax. See [Upgrading from MySQL 4.1 to 5.0](#), for more information about the differences, how they can impact MySQL replication, and what you can do if you encounter such problems.

## Replication and Time Zones

The same system time zone should be set for both master and slave. Otherwise, statements depending on the local time on the master are not replicated properly, such as statements that use the `NOW()` or `FROM_UNIXTIME()` functions. You can set the time zone in which MySQL server runs by using the `--timezone=timezone_name` option of the `mysqld_safe` script or by setting the `TZ` environment variable. See also [Replication and System Functions](#).

If the master is MySQL 4.1 or earlier, both master and slave should also use the same default connection time zone. That is, the `--default-time-zone` parameter should have the same value for both master and slave.

`CONVERT_TZ(..., ..., @@session.time_zone)` is properly replicated only if both master and slave are running MySQL 5.0.4 or newer.

## Replication and Transactions

**Mixing transactional and nontransactional statements within the same transaction.** In general, you should avoid transactions that update both transactional and nontransactional tables in a replication environment. You should also avoid using any statement that accesses both transactional and nontransactional tables and writes to any of them.

In MySQL 5.0 the server uses this rule for binary logging: If the initial statements in a transaction are nontransactional, they are written to the binary log immediately. The remaining statements in the transaction are cached and not written to the binary log until the transaction is committed. (If the transaction is rolled back, the cached statements are written to the binary log only if they make nontransactional changes that cannot be rolled back. Otherwise, they are discarded.)

To apply this rule, the server considers a statement nontransactional if the first changes it makes change nontransactional tables, transactional if the first changes it makes change transactional tables. “First” applies in the sense that a statement may have several effects if it involves such things as triggers, stored functions, or multiple-table updates.

In situations where transactions mix updates to transactional and nontransactional tables, the order of statements in the binary log is correct, and all needed statements are written to the binary log even in case of a `ROLLBACK`. However, when a second connection updates the nontransactional table before the first connection transaction is complete, statements can be logged out of order because the second connection update is written immediately after it is performed, regardless of the state of the transaction being performed by the first connection.

**Using different storage engines on master and slave.** It is possible to replicate transactional tables on the master using nontransactional tables on the slave. For example, you can replicate an `InnoDB` master table as a `MyISAM` slave table. However, if you do this, there are problems if the slave is stopped in the middle of a `BEGIN ... COMMIT` block because the slave restarts at the beginning of the `BEGIN` block.

When the storage engine type of the slave is nontransactional, transactions on the master that mix updates of transactional and nontransactional tables should be avoided because they can cause inconsistency of the data between the master transactional table and the slave nontransactional table. That is, such transactions can lead to master storage engine-specific behavior with the possible effect of replication going out of synchrony. MySQL does not issue a warning about this currently, so extra care should be taken when replicating transactional tables from the master to nontransactional tables on the slaves.

Beginning with MySQL 5.0.56, every transaction (including `autocommit` transactions) is recorded in the binary log as though it starts with a `BEGIN` statement, and ends with either a `COMMIT` or a `ROLLBACK` statement. However, this does *not* apply to nontransactional changes; any statements affecting tables using a nontransactional storage engine such as `MyISAM` are regarded for this purpose as nontransactional, even when `autocommit` is enabled. (Bug #26395)

## Replication and Triggers

**Known issue:** In MySQL 5.0.17, the syntax for `CREATE TRIGGER` changed to include a `DEFINER` clause for specifying which access privileges to check at trigger invocation time. (See `CREATE TRIGGER Syntax`, for more information.) However, if you attempt to replicate from a master server older than MySQL 5.0.17

to a slave running MySQL 5.0.17 through 5.0.19, replication of `CREATE TRIGGER` statements fails on the slave with a `Definer not fully qualified` error. A workaround is to create triggers on the master using a version-specific comment embedded in each `CREATE TRIGGER` statement:

```
CREATE /*!50017 DEFINER = 'root'@'localhost' */ TRIGGER ... ;
```

`CREATE TRIGGER` statements written this way will replicate to newer slaves, which pick up the `DEFINER` clause from the comment and execute successfully.

This slave problem is fixed as of MySQL 5.0.20.

## Replication and Views

Views are always replicated to slaves. Views are filtered by their own name, not by the tables they refer to. This means that a view can be replicated to the slave even if the view contains a table that would normally be filtered out by `replication-ignore-table` rules. Care should therefore be taken to ensure that views do not replicate table data that would normally be filtered for security reasons.

## Replication and Variables

The `foreign_key_checks`, `unique_checks`, and `sql_auto_is_null` variables are all replicated.

`sql_mode` is also replicated except for the `NO_DIR_IN_CREATE` mode. However, when `mysqlbinlog` parses a `SET @@sql_mode = mode` statement, the full `mode` value, including `NO_DIR_IN_CREATE`, is passed to the receiving server.

The `storage_engine` system variable is not replicated, regardless of the logging mode; this is intended to facilitate replication between different storage engines.

The `read_only` system variable is not replicated. In addition, the enabling this variable has different effects with regard to temporary tables, table locking, and the `SET PASSWORD` statement in different MySQL versions.

The `max_heap_table_size` system variable is not replicated. Increasing the value of this variable on the master without doing so on the slave can lead eventually to `Table is full` errors on the slave when trying to execute `INSERT` statements on a `MEMORY` table on the master that is thus permitted to grow larger than its counterpart on the slave. For more information, see [Replication and MEMORY Tables](#).

Starting from MySQL 5.0.3 (master and slave), replication works even if the master and slave have different global character set variables. Starting from MySQL 5.0.4 (master and slave), replication works even if the master and slave have different global time zone variables.

Session variables are not replicated properly when used in statements that update tables. For example, the following sequence of statements will not insert the same data on the master and the slave:

```
SET max_join_size=1000;
INSERT INTO mytable VALUES(@@max_join_size);
```

This does not apply to the common sequence, which replicates correctly as of MySQL 5.0.4.

```
SET time_zone=...;
INSERT INTO mytable VALUES(CONVERT_TZ(..., ..., @@time_zone));
```

Update statements that refer to user-defined variables (that is, variables of the form `@var_name`) are replicated correctly in MySQL 5.0. However, this is not true for versions prior to 4.1. Note that user variable names are case insensitive starting in MySQL 5.0. You should take this into account when setting up replication between MySQL 5.0 and older versions.



In MySQL 5.0.46 and later, the following session variables are written to the binary log and honored by the replication slave when parsing the binary log, regardless of the logging format:

- `sql_mode`
- `foreign_key_checks`
- `unique_checks`
- `character_set_client`
- `collation_connection`
- `collation_database`
- `collation_server`
- `sql_auto_is_null`

**Important**

Even though session variables relating to character sets and collations are written to the binary log, replication between different character sets is not supported.

It is strongly recommended that you always use the same setting for the `lower_case_table_names` system variable on both master and slave. In particular, when a case-sensitive file system is used, and this variable set to 1 on the slave, but to a different value on the master, names of databases are not converted to lowercase, which can cause replication to fail. This is a known issue, which is fixed in MySQL 5.6.

## Replication Compatibility Between MySQL Versions

MySQL supports replication from one major version to the next higher major version. For example, you can replicate from a master running MySQL 4.1 to a slave running MySQL 5.0, from a master running MySQL 5.0 to a slave running MySQL 5.1, and so on.

However, one may encounter difficulties when replicating from an older master to a newer slave if the master uses statements or relies on behavior no longer supported in the version of MySQL used on the slave.

The use of more than 2 MySQL Server versions is not supported in replication setups involving multiple masters, regardless of the number of master or slave MySQL servers. This restriction applies not only to major versions, but to minor versions within the same major version as well. For example, if you are using a chained or circular replication setup, you cannot use MySQL 5.0.21, MySQL 5.0.22, and MySQL 5.0.24 concurrently, although you could use any 2 of these releases together.

In some cases, it is also possible to replicate between a master and a slave that is more than one major version newer than the master. However, there are known issues with trying to replicate from a master running MySQL 4.1 or earlier to a slave running MySQL 5.1 or later. To work around such problems, you can insert a MySQL server running an intermediate version between the two; for example, rather than replicating directly from a MySQL 4.1 master to a MySQL 5.1 slave, it is possible to replicate from a MySQL 4.1 server to a MySQL 5.0 server, and then from the MySQL 5.0 server to a MySQL 5.1 server.

**Important**

It is strongly recommended to use the most recent release available within a given MySQL major version because replication (and other) capabilities are continually

being improved. It is also recommended to upgrade masters and slaves that use early releases of a major version of MySQL to GA (production) releases when the latter become available for that major version.

Replication from newer masters to older slaves may be possible, but is generally not supported. This is due to a number of factors:

- **Binary log format changes.** The binary log format can change between major releases. While we attempt to maintain backward-compatibility, this is not always possible. Major changes were made in MySQL 5.0.3 (for improvements to handling of character sets and `LOAD DATA INFILE`) and 5.0.4 (for improvements to handling of time zones). Because of these changes, replication from a MySQL 5.0.3 or later master to a MySQL 5.0.2 or earlier slave is not supported. This also means that replication from a MySQL 5.0.3 (or later) master to any MySQL 4.1 (or earlier) slave is generally not supported.

This also has significant implications for upgrading replication servers; see [Upgrading a Replication Setup](#), for more information.

- **Use of row-based replication.** Row-based replication was implemented in MySQL 5.1.5, so you cannot replicate using row-based replication from any MySQL 5.0 or later master to a slave older than MySQL 5.1.5.

#### Note

Row-based replication is not available in MySQL 5.0. For more information about row-based replication in MySQL 5.1, see [Replication Formats](#).

- **SQL incompatibilities.** You cannot replicate from a newer master to an older slave using statement-based replication if the statements to be replicated use SQL features available on the master but not on the slave.

For more information on potential replication issues, see [Replication Features and Issues](#).

## Upgrading a Replication Setup

When you upgrade servers that participate in a replication setup, the procedure for upgrading depends on the current server versions and the version to which you are upgrading.

This section applies to upgrading replication from older versions of MySQL to MySQL 5.0. A 4.0 server should be 4.0.3 or newer.

When you upgrade a master to 5.0 from an earlier MySQL release series, you should first ensure that all the slaves of this master are using the same 5.0.x release. If this is not the case, you should first upgrade the slaves. To upgrade each slave, shut it down, upgrade it to the appropriate 5.0.x version, restart it, and restart replication. The 5.0 slave is able to read the old relay logs written prior to the upgrade and to execute the statements they contain. Relay logs created by the slave after the upgrade are in 5.0 format.

After the slaves have been upgraded, shut down the master, upgrade it to the same 5.0.x release as the slaves, and restart it. The 5.0 master is able to read the old binary logs written prior to the upgrade and to send them to the 5.0 slaves. The slaves recognize the old format and handle it properly. Binary logs created by the master subsequent to the upgrade are in 5.0 format. These too are recognized by the 5.0 slaves.

In other words, when upgrading to MySQL 5.0, the slaves must be MySQL 5.0 before you can upgrade the master to 5.0. Note that downgrading from 5.0 to older versions does not work so simply: You must ensure that any 5.0 binary log or relay log has been fully processed, so that you can remove it before proceeding with the downgrade.



Some upgrades may require that you drop and re-create database objects when you move from one MySQL series to the next. For example, collation changes might require that table indexes be rebuilt. Such operations, if necessary, will be detailed at [Upgrading from MySQL 4.1 to 5.0](#). It is safest to perform these operations separately on the slaves and the master, and to disable replication of these operations from the master to the slave. To achieve this, use the following procedure:

1. Stop all the slaves and upgrade them. Restart them with the `--skip-slave-start` [\[33\]](#) option so that they do not connect to the master. Perform any table repair or rebuilding operations needed to re-create database objects, such as use of `REPAIR TABLE` or `ALTER TABLE`, or dumping and reloading tables or triggers.
2. Disable the binary log on the master. To do this without restarting the master, execute a `SET sql_log_bin = 0` statement. Alternatively, stop the master and restart it without the `--log-bin` [\[39\]](#) option. If you restart the master, you might also want to disallow client connections. For example, if all clients connect using TCP/IP, use the `--skip-networking` option when you restart the master.
3. With the binary log disabled, perform any table repair or rebuilding operations needed to re-create database objects. The binary log must be disabled during this step to prevent these operations from being logged and sent to the slaves later.
4. Re-enable the binary log on the master. If you set `sql_log_bin` to 0 earlier, execute a `SET sql_log_bin = 1` statement. If you restarted the master to disable the binary log, restart it with `--log-bin` [\[39\]](#), and without `--skip-networking` so that clients and slaves can connect.
5. Restart the slaves, this time without the `--skip-slave-start` [\[33\]](#) option.

## Troubleshooting Replication

If you have followed the instructions but your replication setup is not working, the first thing to do is *check the error log for messages*. Many users have lost time by not doing this soon enough after encountering problems.

If you cannot tell from the error log what the problem was, try the following techniques:

- Verify that the master has binary logging enabled by issuing a `SHOW MASTER STATUS` statement. If logging is enabled, `Position` is nonzero. If binary logging is not enabled, verify that you are running the master with the `--log-bin` [\[39\]](#) option.
- Verify that the master and slave both were started with the `--server-id` [\[13\]](#) option and that the ID value is unique on each server.
- Verify that the slave is running. Use `SHOW SLAVE STATUS` to check whether the `Slave_IO_Running` and `Slave_SQL_Running` values are both `Yes`. If not, verify the options that were used when starting the slave server. For example, `--skip-slave-start` [\[33\]](#) prevents the slave threads from starting until you issue a `START SLAVE` statement.
- If the slave is running, check whether it established a connection to the master. Use `SHOW PROCESSLIST`, find the I/O and SQL threads and check their `State` column to see what they display. See [Replication Implementation Details](#). If the I/O thread state says `Connecting to master`, check the following:
  - Verify the privileges for the user being used for replication on the master.
  - Check that the host name of the master is correct and that you are using the correct port to connect to the master. The port used for replication is the same as used for client network communication (the default is `3306`). For the host name, ensure that the name resolves to the correct IP address.

- Check that networking has not been disabled on the master or slave. Look for the [skip-networking](#) option in the configuration file. If present, comment it out or remove it.
- If the master has a firewall or IP filtering configuration, ensure that the network port being used for MySQL is not being filtered.
- Check that you can reach the master by using [ping](#) or [traceroute/tracert](#) to reach the host.
- If the slave was running previously but has stopped, the reason usually is that some statement that succeeded on the master failed on the slave. This should never happen if you have taken a proper snapshot of the master, and never modified the data on the slave outside of the slave thread. If the slave stops unexpectedly, it is a bug or you have encountered one of the known replication limitations described in [Replication Features and Issues](#). If it is a bug, see [How to Report Replication Bugs or Problems](#), for instructions on how to report it.
- If a statement that succeeded on the master refuses to run on the slave, try the following procedure if it is not feasible to do a full database resynchronization by deleting the slave's databases and copying a new snapshot from the master:
  1. Determine whether the affected table on the slave is different from the master table. Try to understand how this happened. Then make the slave's table identical to the master's and run [START SLAVE](#).
  2. If the preceding step does not work or does not apply, try to understand whether it would be safe to make the update manually (if needed) and then ignore the next statement from the master.
  3. If you decide that the slave can skip the next statement from the master, issue the following statements:

```
mysql> SET GLOBAL sql_slave_skip_counter = N;  
mysql> START SLAVE;
```

The value of *N* should be 1 if the next statement from the master does not use [AUTO\\_INCREMENT](#) or [LAST\\_INSERT\\_ID\(\)](#). Otherwise, the value should be 2. The reason for using a value of 2 for statements that use [AUTO\\_INCREMENT](#) or [LAST\\_INSERT\\_ID\(\)](#) is that they take two events in the binary log of the master.

See also [SET GLOBAL sql\\_slave\\_skip\\_counter Syntax](#).

4. If you are sure that the slave started out perfectly synchronized with the master, and that no one has updated the tables involved outside of the slave thread, then presumably the discrepancy is the result of a bug. If you are running the most recent version of MySQL, please report the problem. If you are running an older version, try upgrading to the latest production release to determine whether the problem persists.

## How to Report Replication Bugs or Problems

When you have determined that there is no user error involved, and replication still either does not work at all or is unstable, it is time to send us a bug report. We need to obtain as much information as possible from you to be able to track down the bug. Please spend some time and effort in preparing a good bug report.

If you have a repeatable test case that demonstrates the bug, please enter it into our bugs database using the instructions given in [How to Report Bugs or Problems](#). If you have a “phantom” problem (one that you cannot duplicate at will), use the following procedure:

1. Verify that no user error is involved. For example, if you update the slave outside of the slave thread, the data goes out of synchrony, and you can have unique key violations on updates. In this case, the slave thread stops and waits for you to clean up the tables manually to bring them into synchrony. *This is not a replication problem. It is a problem of outside interference causing replication to fail.*
2. Run the slave with the `--log-slave-updates` [22] and `--log-bin` [39] options. These options cause the slave to log the updates that it receives from the master into its own binary logs.
3. Save all evidence before resetting the replication state. If we have no information or only sketchy information, it becomes difficult or impossible for us to track down the problem. The evidence you should collect is:
  - All binary log files from the master
  - All binary log files from the slave
  - The output of `SHOW MASTER STATUS` from the master at the time you discovered the problem
  - The output of `SHOW SLAVE STATUS` from the slave at the time you discovered the problem
  - Error logs from the master and the slave
4. Use `mysqlbinlog` to examine the binary logs. The following should be helpful to find the problem statement. `log_file` and `log_pos` are the `Master_Log_File` and `Read_Master_Log_Pos` values from `SHOW SLAVE STATUS`.

```
shell> mysqlbinlog --start-position=log_pos log_file | head
```

After you have collected the evidence for the problem, try to isolate it as a separate test case first. Then enter the problem with as much information as possible into our bugs database using the instructions at [How to Report Bugs or Problems](#).



---

## Chapter 5. Replication Implementation

### Table of Contents

Replication Implementation Details .....	79
Replication Relay and Status Logs .....	81
The Slave Relay Log .....	81
Slave Status Logs .....	82
How Servers Evaluate Replication Filtering Rules .....	84
Evaluation of Database-Level Replication and Binary Logging Options .....	84
Evaluation of Table-Level Replication Options .....	86
Replication Rule Application .....	90

Replication is based on the master server keeping track of all changes to its databases (updates, deletes, and so on) in its binary log. The binary log serves as a written record of all events that modify database structure or content (data) from the moment the server was started. Typically, `SELECT` statements are not recorded because they modify neither database structure nor content.

Each slave that connects to the master requests a copy of the binary log. That is, it pulls the data from the master, rather than the master pushing the data to the slave. The slave also executes the events from the binary log that it receives. This has the effect of repeating the original changes just as they were made on the master. Tables are created or their structure modified, and data is inserted, deleted, and updated according to the changes that were originally made on the master.

Because each slave is independent, the replaying of the changes from the master's binary log occurs independently on each slave that is connected to the master. In addition, because each slave receives a copy of the binary log only by requesting it from the master, the slave is able to read and update the copy of the database at its own pace and can start and stop the replication process at will without affecting the ability to update to the latest database status on either the master or slave side.

For more information on the specifics of the replication implementation, see [Replication Implementation Details](#).

Masters and slaves report their status in respect of the replication process regularly so that you can monitor them. See [Examining Thread Information](#), for descriptions of all replicated-related states.

The master binary log is written to a local relay log on the slave before it is processed. The slave also records information about the current position with the master's binary log and the local relay log. See [Replication Relay and Status Logs](#).

Database changes are filtered on the slave according to a set of rules that are applied according to the various configuration options and variables that control event evaluation. For details on how these rules are applied, see [How Servers Evaluate Replication Filtering Rules](#).

### Replication Implementation Details

MySQL replication capabilities are implemented using three threads, one on the master server and two on the slave:

- **Binlog dump thread.** The master creates a thread to send the binary log contents to a slave when the slave connects. This thread can be identified in the output of `SHOW PROCESSLIST` on the master as the `Binlog Dump` thread.

The binlog dump thread acquires a lock on the master's binary log for reading each event that is to be sent to the slave. As soon as the event has been read, the lock is released, even before the event is sent to the slave.

- **Slave I/O thread.** When a `START SLAVE` statement is issued on a slave server, the slave creates an I/O thread, which connects to the master and asks it to send the updates recorded in its binary logs.

The slave I/O thread reads the updates that the master's `Binlog Dump` thread sends (see previous item) and copies them to local files that comprise the slave's relay log.

The state of this thread is shown as `Slave_IO_running` in the output of `SHOW SLAVE STATUS` or as `Slave_running` in the output of `SHOW STATUS`.

- **Slave SQL thread.** The slave creates an SQL thread to read the relay log that is written by the slave I/O thread and execute the events contained therein.

In the preceding description, there are three threads per master/slave connection. A master that has multiple slaves creates one binlog dump thread for each currently connected slave, and each slave has its own I/O and SQL threads.

A slave uses two threads to separate reading updates from the master and executing them into independent tasks. Thus, the task of reading statements is not slowed down if statement execution is slow. For example, if the slave server has not been running for a while, its I/O thread can quickly fetch all the binary log contents from the master when the slave starts, even if the SQL thread lags far behind. If the slave stops before the SQL thread has executed all the fetched statements, the I/O thread has at least fetched everything so that a safe copy of the statements is stored locally in the slave's relay logs, ready for execution the next time that the slave starts. This enables the master server to purge its binary logs sooner because it no longer needs to wait for the slave to fetch their contents.

The `SHOW PROCESSLIST` statement provides information that tells you what is happening on the master and on the slave regarding replication. For information on master states, see [Replication Master Thread States](#). For slave states, see [Replication Slave I/O Thread States](#), and [Replication Slave SQL Thread States](#).

The following example illustrates how the three threads show up in the output from `SHOW PROCESSLIST`.

On the master server, the output from `SHOW PROCESSLIST` looks like this:

```
mysql> SHOW PROCESSLIST\G
***** 1. row *****
      Id: 2
     User: root
    Host: localhost:32931
       db: NULL
Command: Binlog Dump
      Time: 94
     State: Has sent all binlog to slave; waiting for binlog to
           be updated
      Info: NULL
```

Here, thread 2 is a `Binlog Dump` replication thread that services a connected slave. The `State` information indicates that all outstanding updates have been sent to the slave and that the master is waiting for more updates to occur. If you see no `Binlog Dump` threads on a master server, this means that replication is not running; that is, no slaves are currently connected.

On a slave server, the output from `SHOW PROCESSLIST` looks like this:

```
mysql> SHOW PROCESSLIST\G
***** 1. row *****
```

```

    Id: 10
    User: system user
    Host:
    db: NULL
Command: Connect
    Time: 11
    State: Waiting for master to send event
    Info: NULL
***** 2. row *****
    Id: 11
    User: system user
    Host:
    db: NULL
Command: Connect
    Time: 11
    State: Has read all relay log; waiting for the slave I/O
          thread to update it
    Info: NULL

```

The `State` information indicates that thread 10 is the I/O thread that is communicating with the master server, and thread 11 is the SQL thread that is processing the updates stored in the relay logs. At the time that `SHOW PROCESSLIST` was run, both threads were idle, waiting for further updates.

The value in the `Time` column can show how late the slave is compared to the master. See [MySQL 5.0 FAQ: Replication](#). If sufficient time elapses on the master side without activity on the `Binlog Dump` thread, the master determines that the slave is no longer connected. As for any other client connection, the timeouts for this depend on the values of `net_write_timeout` and `net_retry_count`; for more information about these, see [Server System Variables](#).

The `SHOW SLAVE STATUS` statement provides additional information about replication processing on a slave server. See [Checking Replication Status](#).

## Replication Relay and Status Logs

During replication, a slave server creates several logs that hold the binary log events relayed from the master to the slave, and to record information about the current status and location within the relay log. There are three types of logs used in the process, listed here:

- The *relay log* consists of the events read from the binary log of the master and written by the slave I/O thread. Events in the relay log are executed on the slave as part of the SQL thread.
- The *master info log* contains status and current configuration information for the slave's connection to the master. This log holds information on the master host name, login credentials, and coordinates indicating how far the slave has read from the master's binary log.
- The *relay log info log* holds status information about the execution point within the slave's relay log.

## The Slave Relay Log

The relay log, like the binary log, consists of a set of numbered files containing events that describe database changes, and an index file that contains the names of all used relay log files.

The term “relay log file” generally denotes an individual numbered file containing database events. The term “relay log” collectively denotes the set of numbered relay log files plus the index file.

Relay log files have the same format as binary log files and can be read using `mysqlbinlog` (see [mysqlbinlog — Utility for Processing Binary Log Files](#)).

By default, relay log file names have the form `host_name-relay-bin.nnnnnn` in the data directory, where `host_name` is the name of the slave server host and `nnnnnn` is a sequence number. Successive

relay log files are created using successive sequence numbers, beginning with `000001`. The slave uses an index file to track the relay log files currently in use. The default relay log index file name is `host_name-relay-bin.index` in the data directory.

The default relay log file and relay log index file names can be overridden with, respectively, the `--relay-log` [26] and `--relay-log-index` [26] server options (see [Replication and Binary Logging Options and Variables](#)).

If a slave uses the default host-based relay log file names, changing a slave's host name after replication has been set up can cause replication to fail with the errors `Failed to open the relay log` and `Could not find target log during relay log initialization`. This is a known issue (see Bug #2122). If you anticipate that a slave's host name might change in the future (for example, if networking is set up on the slave such that its host name can be modified using DHCP), you can avoid this issue entirely by using the `--relay-log` [26] and `--relay-log-index` [26] options to specify relay log file names explicitly when you initially set up the slave. This will make the names independent of server host name changes.

If you encounter the issue after replication has already begun, one way to work around it is to stop the slave server, prepend the contents of the old relay log index file to the new one, and then restart the slave. On a Unix system, this can be done as shown here:

```
shell> cat new_relay_log_name.index >> old_relay_log_name.index
shell> mv old_relay_log_name.index new_relay_log_name.index
```

A slave server creates a new relay log file under the following conditions:

- Each time the I/O thread starts.
- When the logs are flushed; for example, with `FLUSH LOGS` or `mysqladmin flush-logs`.
- When the size of the current relay log file becomes “too large,” determined as follows:
  - If the value of `max_relay_log_size` is greater than 0, that is the maximum relay log file size.
  - If the value of `max_relay_log_size` is 0, `max_binlog_size` [43] determines the maximum relay log file size.

The SQL thread automatically deletes each relay log file as soon as it has executed all events in the file and no longer needs it. There is no explicit mechanism for deleting relay logs because the SQL thread takes care of doing so. However, `FLUSH LOGS` rotates relay logs, which influences when the SQL thread deletes them.

## Slave Status Logs

A replication slave server creates two logs. By default, these logs are files named `master.info` and `relay-log.info` and created in the data directory. The names and locations of these files can be changed by using the `--master-info-file` [24] and `--relay-log-info-file` [27] options, respectively. See [Replication and Binary Logging Options and Variables](#).

The two status logs contain information like that shown in the output of the `SHOW SLAVE STATUS` statement, which is discussed in [SQL Statements for Controlling Slave Servers](#). Because the status logs are stored on disk, they survive a slave server's shutdown. The next time the slave starts up, it reads the two logs to determine how far it has proceeded in reading binary logs from the master and in processing its own relay logs.

The master info log should be protected because it contains the password for connecting to the master. See [Passwords and Logging](#).



The slave I/O thread updates the master info log. The following table shows the correspondence between the lines in the `master.info` file and the columns displayed by `SHOW SLAVE STATUS`.

Line in <code>master.info</code>	<code>SHOW SLAVE STATUS</code> Column	Description
1		Number of lines in the file
2	<code>Master_Log_File</code>	The name of the master binary log currently being read from the master
3	<code>Read_Master_Log_Pos</code>	The current position within the master binary log that have been read from the master
4	<code>Master_Host</code>	The host name of the master
5	<code>Master_User</code>	The user name used to connect to the master
6	Password (not shown by <code>SHOW SLAVE STATUS</code> )	The password used to connect to the master
7	<code>Master_Port</code>	The network port used to connect to the master
8	<code>Connect_Retry</code>	The period (in seconds) that the slave will wait before trying to reconnect to the master
9	<code>Master_SSL_Allowed</code>	Indicates whether the server supports SSL connections
10	<code>Master_SSL_CA_File</code>	The file used for the Certificate Authority (CA) certificate
11	<code>Master_SSL_CA_Path</code>	The path to the Certificate Authority (CA) certificates
12	<code>Master_SSL_Cert</code>	The name of the SSL certificate file
13	<code>Master_SSL_Cipher</code>	The list of possible ciphers used in the handshake for the SSL connection
14	<code>Master_SSL_Key</code>	The name of the SSL key file

The slave SQL thread updates the relay log info log. The following table shows the correspondence between the lines in the `relay-log.info` file and the columns displayed by `SHOW SLAVE STATUS`.

Line in <code>relay-log.info</code>	<code>SHOW SLAVE STATUS</code> Column	Description
1	<code>Relay_Log_File</code>	The name of the current relay log file
2	<code>Relay_Log_Pos</code>	The current position within the relay log file; events up to this position have been executed on the slave database
3	<code>Relay_Master_Log_File</code>	The name of the master binary log file from which the events in the relay log file were read
4	<code>Exec_Master_Log_Pos</code>	The equivalent position within the master's binary log file of events that have already been executed

The contents of the `relay-log.info` file and the states shown by the `SHOW SLAVE STATUS` statement might not match if the `relay-log.info` file has not been flushed to disk. Ideally, you should only view `relay-log.info` on a slave that is offline (that is, `mysqld` is not running). For a running system, `SHOW SLAVE STATUS` should be used.

When you back up the slave's data, you should back up these two status logs, along with the relay log files. The status logs are needed to resume replication after you restore the data from the slave. If you lose the relay logs but still have the relay log info log, you can check it to determine how far the SQL thread has executed in the master binary logs. Then you can use `CHANGE MASTER TO` with the `MASTER_LOG_FILE` and `MASTER_LOG_POS` options to tell the slave to re-read the binary logs from that point. Of course, this requires that the binary logs still exist on the master.

## How Servers Evaluate Replication Filtering Rules

If a master server does not write a statement to its binary log, the statement is not replicated. If the server does log the statement, the statement is sent to all slaves and each slave determines whether to execute it or ignore it.

On the master, you can control which databases to log changes for by using the `--binlog-do-db` [41] and `--binlog-ignore-db` [42] options to control binary logging. For a description of the rules that servers use in evaluating these options, see [Evaluation of Database-Level Replication and Binary Logging Options](#). You should not use these options to control which databases and tables are replicated. Instead, use filtering on the slave to control the events that are executed on the slave.

On the slave side, decisions about whether to execute or ignore statements received from the master are made according to the `--replicate-*` options that the slave was started with. (See [Replication and Binary Logging Options and Variables](#).)

In the simplest case, when there are no `--replicate-*` options, the slave executes all statements that it receives from the master. Otherwise, the result depends on the particular options given.

Database-level options (`--replicate-do-db` [28], `--replicate-ignore-db` [28]) are checked first; see [Evaluation of Database-Level Replication and Binary Logging Options](#), for a description of this process. If no matching database-level options are found, option checking proceeds to any table-level options that may be in use, as discussed in [Evaluation of Table-Level Replication Options](#).

To make it easier to determine what effect an option set will have, it is recommended that you avoid mixing “do” and “ignore” options, or wildcard and nonwildcard options. An example of the latter that may have unintended effects is the use of `--replicate-do-db` [28] and `--replicate-wild-do-table` [30] together, where `--replicate-wild-do-table` [30] uses a pattern for the database name that matches the name given for `--replicate-do-db` [28]. Suppose a replication slave is started with `--replicate-do-db=dbx` [28] `--replicate-wild-do-table=db%.t1` [30]. Then, suppose that on the master, you issue the statement `CREATE DATABASE dbx`. Although you might expect it, this statement is not replicated because it does not reference a table named `t1`.

If any `--replicate-rewrite-db` [30] options were specified, they are applied before the `--replicate-*` filtering rules are tested.

### Note

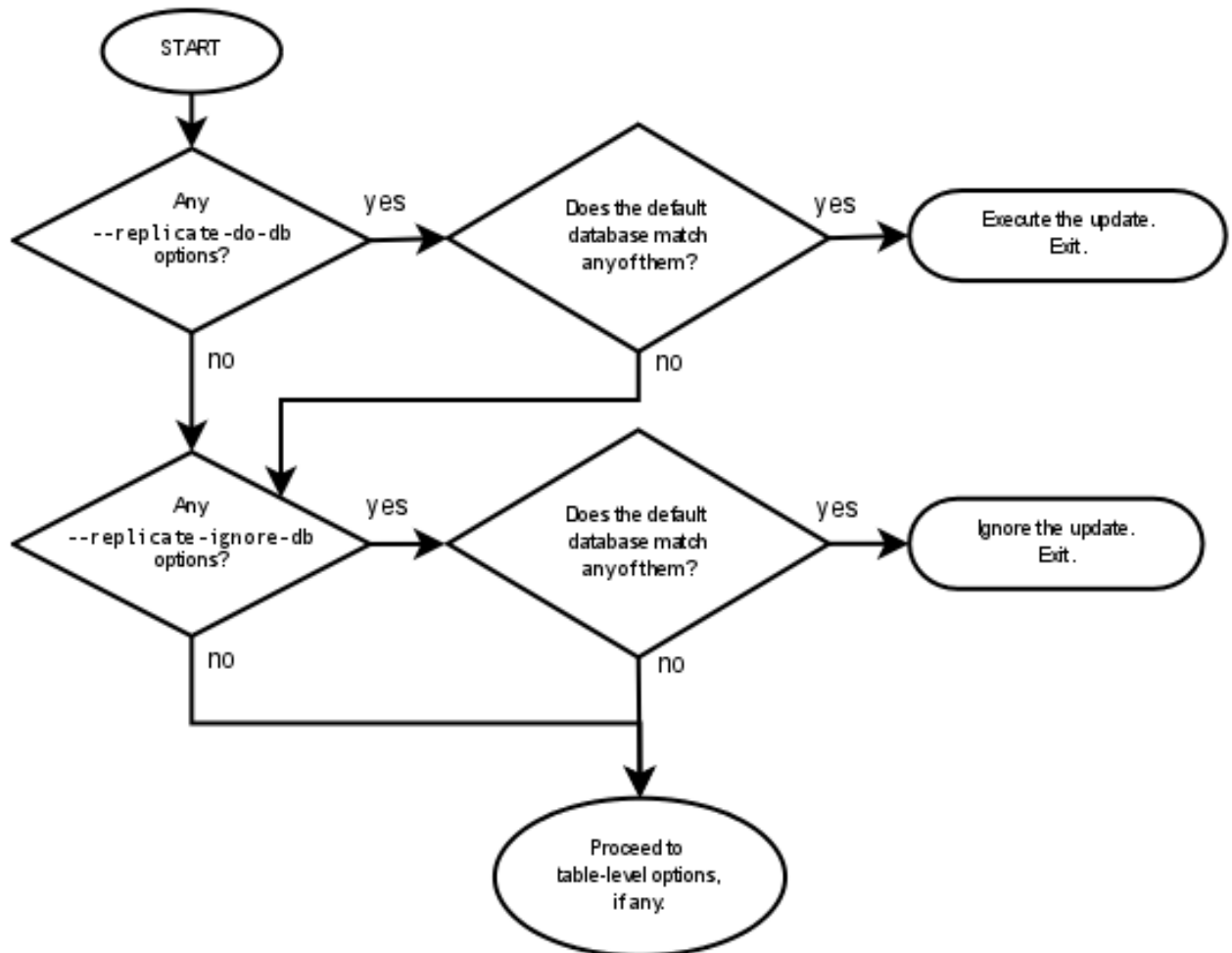
In MySQL 5.0, database-level filtering options are case-sensitive on platforms supporting case sensitivity in filenames, whereas table-level filtering options are not (regardless of platform). This is true regardless of the value of the `lower_case_table_names` system variable.

## Evaluation of Database-Level Replication and Binary Logging Options

When evaluating replication options, the slave begins by checking to see whether there are any `--replicate-do-db` [28] or `--replicate-ignore-db` [28] options that apply. When using `--binlog-`

`do-db` [41] or `--binlog-ignore-db` [42], the process is similar, but the options are checked on the master.

Checking of the database-level options proceeds as shown in the following diagram.



The steps involved are listed here:

1. Are there any `--replicate-do-db` [28] options?
  - **Yes.** Do any of them match the database?
    - **Yes.** Execute the statement and exit.
    - **No.** Continue to step 2.
  - **No.** Continue to step 2.
2. Are there any `--replicate-ignore-db` [28] options?
  - **Yes.** Do any of them match the database?
    - **Yes.** Ignore the statement and exit.
    - **No.** Continue to step 3.

- **No.** Continue to step 3.
3. Proceed to checking the table-level replication options, if there are any. For a description of how these options are checked, see [Evaluation of Table-Level Replication Options](#).

**Important**

A statement that is still permitted at this stage is not yet actually executed. The statement is not executed until all table-level options (if any) have also been checked, and the outcome of that process permits execution of the statement.

For binary logging, the steps involved are listed here:

1. Are there any `--binlog-do-db` [41] or `--binlog-ignore-db` [42] options?
  - **Yes.** Continue to step 2.
  - **No.** Log the statement and exit.
2. Is there a default database (has any database been selected by `USE`)?
  - **Yes.** Continue to step 3.
  - **No.** Ignore the statement and exit.
3. There is a default database. Are there any `--binlog-do-db` [41] options?
  - **Yes.** Do any of them match the database?
    - **Yes.** Log the statement and exit.
    - **No.** Ignore the statement and exit.
  - **No.** Continue to step 4.
4. Do any of the `--binlog-ignore-db` [42] options match the database?
  - **Yes.** Ignore the statement and exit.
  - **No.** Log the statement and exit.

**Important**

An exception is made in the rules just given for the `CREATE DATABASE`, `ALTER DATABASE`, and `DROP DATABASE` statements. In those cases, the database being *created, altered, or dropped* replaces the default database when determining whether to log or to ignore updates.

`--binlog-do-db` [41] can sometimes mean “ignore other databases”. For example, a server running with only `--binlog-do-db=sales` [41] does not write to the binary log statements for which the default database differs from `sales`.

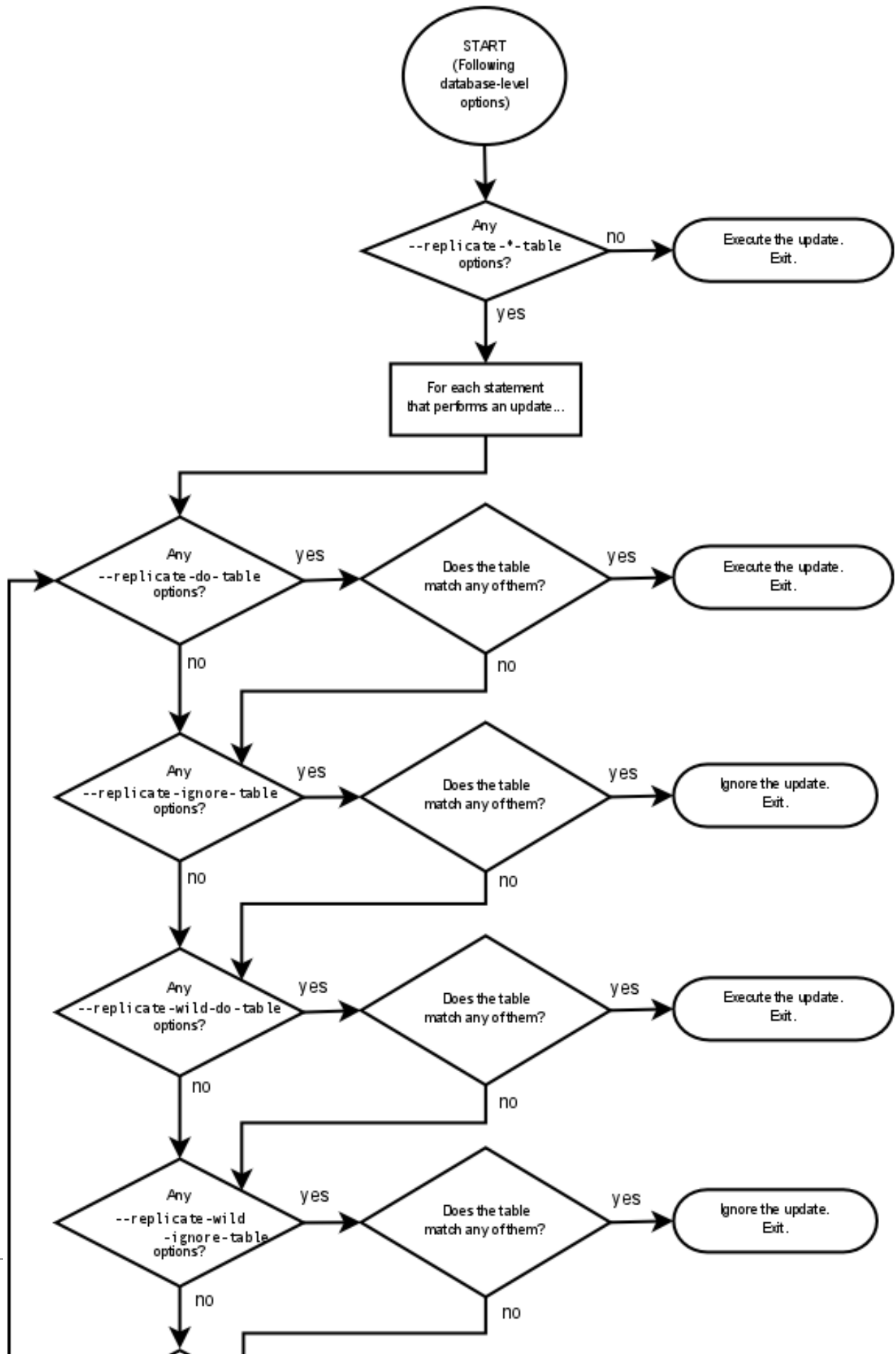
Relay log files have the same format as binary log files and can be read using `mysqlbinlog`.

## Evaluation of Table-Level Replication Options

The slave checks for and evaluates table options only if no matching database options were found (see [Evaluation of Database-Level Replication and Binary Logging Options](#)).

First, as a preliminary condition, the slave checks whether the statement occurs within a stored function, in which case the slave executes the statement and exits.

Having reached this point, if there are no table options, the slave simply executes all statements. If there are any `--replicate-do-table` [29] or `--replicate-wild-do-table` [30] options, the statement must match one of these if it is to be executed; otherwise, it is ignored. If there are any `--replicate-ignore-table` [29] or `--replicate-wild-ignore-table` [31] options, all statements are executed except those that match any of these options. This process is illustrated in the following diagram.



The `master.info` file should be protected because it contains the password for connecting to the master. See [Passwords and Logging](#).

The following steps describe this evaluation in more detail:

1. Are there any table options?
  - **Yes.** Continue to step 2.
  - **No.** Execute the statement and exit.
2. Are there any `--replicate-do-table [29]` options?
  - **Yes.** Does the table match any of them?
    - **Yes.** Execute the statement and exit.
    - **No.** Continue to step 3.
  - **No.** Continue to step 3.
3. Are there any `--replicate-ignore-table [29]` options?
  - **Yes.** Does the table match any of them?
    - **Yes.** Ignore the statement and exit.
    - **No.** Continue to step 4.
  - **No.** Continue to step 4.
4. Are there any `--replicate-wild-do-table [30]` options?
  - **Yes.** Does the table match any of them?
    - **Yes.** Execute the statement and exit.
    - **No.** Continue to step 5.
  - **No.** Continue to step 5.
5. Are there any `--replicate-wild-ignore-table [31]` options?
  - **Yes.** Does the table match any of them?
    - **Yes.** Ignore the statement and exit.
    - **No.** Continue to step 6.
  - **No.** Continue to step 6.
6. Are there any `--replicate-do-table [29]` or `--replicate-wild-do-table [30]` options?
  - **Yes.** Ignore the statement and exit.
  - **No.** Execute the statement and exit.

## Replication Rule Application

This section provides additional explanation and examples of usage for different combinations of replication filtering options.

Some typical combinations of replication filter rule types are given in the following table:

Condition (Types of Options)	Outcome
No <code>--replicate-*</code> options at all:	The slave executes all events that it receives from the master.
<code>--replicate-*-db</code> options, but no table options:	The slave accepts or ignores statements using the database options. It executes all statements permitted by those options because there are no table restrictions.
<code>--replicate-*-table</code> options, but no database options:	All statements are accepted at the database-checking stage because there are no database conditions. The slave executes or ignores statements based solely on the table options.
A combination of database and table options:	The slave accepts or ignores statements using the database options. Then it evaluates all statements permitted by those options according to the table options. This can sometimes lead to results that seem counterintuitive; see the text for an example.

A more complex example follows.

Suppose that we have two tables `mytbl1` in database `db1` and `mytbl2` in database `db2` on the master, and the slave is running with the following options (and no other replication filtering options):

```
replicate-ignore-db = db1
replicate-do-table  = db2.tbl2
```

Now we execute the following statements on the master:

```
USE db1;
INSERT INTO db2.tbl2 VALUES (1);
```

The outcome may not match initial expectations, because the `USE` statement causes `db1` to be the default database. Thus the `--replicate-ignore-db` [28] option matches, which causes the `INSERT` statement to be ignored. Because there was a match with a database-level option, the table options are not checked; processing immediately moves to the next statement executed on the master.