16.1.6.3 Replication Slave Options and Variables

This section explains the server options and system variables that apply to slave replication servers and contains the following:

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. Specify system variable values using `SET`.

Server ID. On the master and each slave, you must use the `server-id` 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
```

Startup Options for Replication Slaves

This section explains 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.

- `--log-slave-updates`

Command-Line Format	`--log-slave-updates`	
System Variable	**Name**	`log_slave_updates`
	Variable Scope	Global
	Dynamic Variable	No
Permitted Values	**Type**	boolean
	Default	`OFF`

  Normally, a slave does not write any updates that are received from a master server to its own binary log. This option causes the slave to write 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` option to enable binary logging. `--log-slave-updates` 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` to enable binary logging, and `B` with the `--log-slave-updates` option so that updates received from `A` are logged by `B` to its binary log.

- `--log-slow-slave-statements`

Removed	5.7.1	
Command-Line Format	`--log-slow-slave-statements` (5.7.0)	
Permitted Values	**Type**	boolean
	Default	`OFF`

When the slow query log is enabled, this option enables logging for queries that have taken more than `long_query_time` seconds to execute on the slave.

This command-line option was removed in MySQL 5.7.1 and replaced by the `log_slow_slave_statements` system variable. The system variable can be set on the command line or in option files the same way as the option, so there is no need for any changes at server startup, but the system variable also makes it possible to examine or set the value at runtime.

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

Deprecated	5.7.2	
Command-Line Format	`--log-warnings[=#]`	
System Variable	**Name**	`log_warnings`
	Variable Scope	Global
	Dynamic Variable	Yes
Permitted Values (32-bit platforms, <= 5.7.1)	**Type**	integer
	Default	`1`
	Min Value	`0`
	Max Value	`4294967295`
Permitted Values (32-bit platforms, >= 5.7.2)	**Type**	integer
	Default	`2`
	Min Value	`0`
	Max Value	`4294967295`
Permitted Values (64-bit platforms, <= 5.7.1)	**Type**	integer
	Default	`1`
	Min Value	`0`
	Max Value	`18446744073709551615`
Permitted Values (64-bit platforms, >= 5.7.2)	**Type**	integer
	Default	`2`
	Min Value	`0`
	Max Value	`18446744073709551615`

> **Note**
>
> As of MySQL 5.7.2, the `log_error_verbosity` system variable is preferred over, and should be used instead of, the `--log-warnings` option or `log_warnings` system variable. For more information, see the descriptions of `log_error_verbosity` and `log_warnings`. The `--log-warnings` command-line option and `log_warnings` system variable are deprecated and will be removed in a future MySQL release.

Causes the server to record 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 or connection failure, and provides information about how each slave thread started. This variable is enabled by default (the default is 1 before MySQL 5.7.2, 2 as of 5.7.2). To disable it, set it to 0. The server logs messages about statements that are unsafe for statement-based logging if the value is greater than 0. Aborted connections and access-denied errors for new connection attempts are logged if the value is greater than 1. See Section B.5.2.11, "Communication Errors and Aborted Connections".

> **Note**
>
> The effects of this option are not limited to replication. It produces warnings across a spectrum of server activities.

- `--master-info-file=`***`file_name`***

Command-Line Format	`--master-info-file=file_name`	
Permitted Values	**Type**	file name
	Default	`master.info`

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 Section 16.2.4.2, "Slave Status Logs".

- `--master-retry-count=`***`count`***

Deprecated	5.6.1	
Command-Line Format	`--master-retry-count=#`	
Permitted Values (32-bit platforms)	**Type**	integer
	Default	`86400`
	Min Value	`0`
	Max Value	`4294967295`
Permitted Values (64-bit platforms)	**Type**	integer
	Default	`86400`

	Min Value	`0`
	Max Value	`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` option of the `CHANGE MASTER TO` statement (default 60). Reconnects are triggered when data reads by the slave time out according to the `--slave-net-timeout` option. The default value is 86400. A value of 0 means "infinite"; the slave attempts to connect forever.

This option is deprecated and will be removed in a future MySQL release. Applications should be updated to use the `MASTER_RETRY_COUNT` option of the `CHANGE MASTER TO` statement instead.

- `--max-relay-log-size=`***`size`***

Command-Line Format	`--max-relay-log-size=#`	
System Variable	**Name**	`max_relay_log_size`
	Variable Scope	Global
	Dynamic Variable	Yes
Permitted Values	**Type**	integer
	Default	`0`
	Min Value	`0`
	Max Value	`1073741824`

The size at which the server rotates relay log files automatically. If this value is nonzero, the relay log is rotated automatically when its size exceeds this value. If this value is zero (the default), the size at which relay log rotation occurs is determined by the value of `max_binlog_size`. For more information, see Section 16.2.4.1, "The Slave Relay Log".

- `--relay-log=`***`file_name`***

Command-Line Format	`--relay-log=file_name`	
System Variable	**Name**	`relay_log`
	Variable Scope	Global
	Dynamic Variable	No
Permitted Values	**Type**	file name

The base name for the relay log. For the default replication channel, the default base name for relay logs is ***`host_name`***`-relay-bin`. For non-default replication channels, the default base name for relay logs is ***`host_name`***`-`***`channel`***`-relay-bin`, where ***`channel`*** is the name of the replication channel recorded in this relay log. The server writes the file in the data directory unless the base name 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 base name.

Due to the manner in which MySQL parses server options, if you specify this option, you must supply a value; *the default base name is used only if the option is not actually specified*. If you use the `--relay-log` 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 Section 4.2.3, “Specifying Program Options”.

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

When the server reads an entry from the index file, it checks whether the entry contains a relative path. If it does, the relative part of the path is replaced with the absolute path set using the `--relay-log` option. An absolute path remains unchanged; in such a case, the index must be edited manually to enable the new path or paths to be used. Previously, manual intervention was required whenever relocating the binary log or relay log files. (Bug #11745230, Bug #12133)

You may find the `--relay-log` 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.

You can obtain the relay log file name (and path) from the `relay_log_basename` system variable.

- `--relay-log-index=`***`file_name`***

Command-Line Format	`--relay-log-index=file_name`	
System Variable	Name	`relay_log_index`
	Variable Scope	Global
	Dynamic Variable	No
Permitted Values	Type	file name

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 server. For the default replication channel, the default name is ***`host_name`***`-relay-bin.index`. For non-default replication channels, the default name is ***`host_name-channel`***`-relay-bin.index`, where ***`channel`*** is the name of the replication channel recorded in this relay log index.

Due to the manner in which MySQL parses server options, if you specify this option, you must supply a value; *the default base name is used only if the option is not actually specified*. If you use the `--relay-log-index` 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 Section 4.2.3, “Specifying Program Options”.

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

- `--relay-log-info-file=`***`file_name`***

Command-Line Format	`--relay-log-info-file=file_name`	
Permitted Values	**Type**	file name
	Default	`relay-log.info`

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 Section 16.2.4.2, “Slave Status Logs”.

- `--relay-log-purge={0|1}`

Command-Line Format	`--relay-log-purge`	
System Variable	**Name**	`relay_log_purge`
	Variable Scope	Global
	Dynamic Variable	Yes
Permitted Values	**Type**	boolean
	Default	`TRUE`

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`***. Disabling purging of relay logs when using the `--relay-log-recovery` option puts data consistency at risk.

- `--relay-log-recovery={0|1}`

Command-Line Format	`--relay-log-recovery`	
Permitted Values	**Type**	boolean
	Default	`FALSE`

Enables automatic relay log recovery immediately following server startup. The recovery process creates a new relay log file, initializes the SQL thread position to this new relay log, and initializes the I/O thread to the SQL thread position. Reading of the relay log from the master then continues. This should be used following an unexpected halt of a replication slave to ensure that no possibly corrupted relay logs are processed. The default value is 0 (disabled).

This variable can be set to 1 to make a slave resilient to unexpected halts, see Section 16.3.2, “Handling an Unexpected Halt of a Replication Slave” for more information. Enabling the `--relay-log-recovery` option when `relay-log-purge` is disabled risks reading the relay log from files that were not purged, leading to data inconsistency.

When using a multi-threaded slave (in other words `slave_parallel_workers` is greater than 0), inconsistencies such as gaps can occur in the sequence of transactions that have been executed from the relay log. Enabling the `--relay-log-recovery` option when there are inconsistencies causes an error and the option has no effect. The solution in this situation is to issue `START SLAVE UNTIL SQL_AFTER_MTS_GAPS`, which brings the server to a more consistent state, then issue `RESET SLAVE` to remove the relay logs. See Section 16.4.1.34, "Replication and Transaction Inconsistencies" for more information.

- `--relay-log-space-limit=`***`size`***

Command-Line Format	`--relay-log-space-limit=#`	
System Variable	**Name**	`relay_log_space_limit`
	Variable Scope	Global
	Dynamic Variable	No
Permitted Values (32-bit platforms)	**Type**	integer
	Default	`0`
	Min Value	`0`
	Max Value	`4294967295`
Permitted Values (64-bit platforms)	**Type**	integer
	Default	`0`
	Min Value	`0`
	Max Value	`18446744073709551615`

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` to less than twice the value of `--max-relay-log-size` (or `--max-binlog-size` if `--max-relay-log-size` is 0). In that case, there is a chance that the I/O thread waits for free space because `--relay-log-space-limit` 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` temporarily.

- `--replicate-do-db=`***`db_name`***

Command-Line Format	`--replicate-do-db=name`	
Permitted Values	**Type**	string

Creates a replication filter using the name of a database. In MySQL 5.7.3 and later, such filters can also be created using `CHANGE REPLICATION FILTER REPLICATE_DO_DB`. The precise effect of this filtering depends on whether statement-based or row-based replication is in use, and are described in the next several paragraphs.

Statement-based replication. 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; however, doing so does *not* replicate cross-database statements such as `UPDATE` ***`some_db.some_table`*** `SET foo='bar'` while a different database (or no database) is selected.

> **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 when using statement-based replication: If the slave is started with `--replicate-do-db=sales` 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` 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.

Row-based replication. Tells the slave SQL thread to restrict replication to database ***`db_name`***. Only tables belonging to ***`db_name`*** are changed; the current database has no effect on this. Suppose that the slave is started with `--replicate-do-db=sales` and row-based replication is in effect, and then the following statements are run on the master:

```
USE prices;
UPDATE sales.february SET amount=amount+100;
```

The `february` table in the `sales` database on the slave is changed in accordance with the `UPDATE` statement; this occurs whether or not the `USE` statement was issued. However, issuing the following statements on the master has no effect on the slave when using row-based replication and `--replicate-do-db=sales`:

```
USE prices;
UPDATE prices.march SET amount=amount-25;
```

Even if the statement `USE prices` were changed to `USE sales`, the `UPDATE` statement's effects would still not be replicated.

Another important difference in how `--replicate-do-db` is handled in statement-based replication as opposed to row-based replication occurs with regard to statements that refer to multiple databases. Suppose that the slave is started with `--replicate-do-db=db1`, and the following statements are executed on the master:

```
USE db1;
UPDATE db1.table1 SET col1 = 10, db2.table2 SET col2 = 20;
```

If you are using statement-based replication, then both tables are updated on the slave. However, when using row-based replication, only `table1` is affected on the slave; since `table2` is in a different database, `table2` on the slave is not changed by the `UPDATE`. Now suppose that, instead of the `USE db1` statement, a `USE db4` statement had been used:

```
USE db4;
UPDATE db1.table1 SET col1 = 10, db2.table2 SET col2 = 20;
```

In this case, the `UPDATE` statement would have no effect on the slave when using statement-based replication. However, if you are using row-based replication, the `UPDATE` would change `table1` on the slave, but not `table2`—in other words, only tables in the database named by `--replicate-do-db` are changed, and the choice of default database has no effect on this behavior.

If you need cross-database updates to work, use `--replicate-wild-do-table=`***`db_name`***`.%` instead. See Section 16.2.5, “How Servers Evaluate Replication Filtering Rules”.

> ### Note
>
> This option affects replication in the same manner that `--binlog-do-db` affects binary logging, and the effects of the replication format on how `--replicate-do-db` affects replication behavior are the same as those of the logging format on the behavior of `--binlog-do-db`.
>
> This option has no effect on `BEGIN`, `COMMIT`, or `ROLLBACK` statements.

- `--replicate-ignore-db=`***`db_name`***

Command-Line Format	`--replicate-ignore-db=name`	
Permitted Values	**Type**	string

Creates a replication filter using the name of a database. In MySQL 5.7.3 and later, such filters can also be created using `CHANGE REPLICATION FILTER REPLICATE_IGNORE_DB`. As with `--replicate-do-db`, the precise effect of this filtering depends on whether statement-based or row-based replication is in use, and are described in the next several paragraphs.

Statement-based replication. Tells the slave SQL thread not to replicate any statement where the default database (that is, the one selected by `USE`) is ***`db_name`***.

Row-based replication. Tells the slave SQL thread not to update any tables in the database ***`db_name`***. The default database has no effect.

When using statement-based replication, the following example does not work as you might expect. Suppose that the slave is started with `--replicate-ignore-db=sales` and you issue the following statements on the master:

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

The `UPDATE` statement *is* replicated in such a case because `--replicate-ignore-db` applies only to the default database (determined by the `USE` statement). Because the `sales` database was specified explicitly in the statement, the statement has not been filtered. However, when using row-based replication, the `UPDATE` statement's effects are *not* propagated to the slave, and the slave's copy of the `sales.january` table is unchanged; in this instance, `--replicate-ignore-db=sales` causes *all* changes made to tables in the master's copy of the `sales` database to be ignored by the slave.

To specify more than one database to ignore, use this option multiple times, once for each database. 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.

You should not use this option if you are using cross-database updates and you do not want these updates to be replicated. See Section 16.2.5, “How Servers Evaluate Replication Filtering Rules”.

If you need cross-database updates to work, use `--replicate-wild-ignore-table=`***`db_name`***`.%` instead. See Section 16.2.5, “How Servers Evaluate Replication Filtering Rules”.

> **Note**
>
> This option affects replication in the same manner that `--binlog-ignore-db` affects binary logging, and the effects of the replication format on how `--replicate-ignore-db` affects replication behavior are the same as those of the logging format on the behavior of `--binlog-ignore-db`.
>
> This option has no effect on `BEGIN`, `COMMIT`, or `ROLLBACK` statements.

- `--replicate-do-table=`***`db_name.tbl_name`***

Command-Line Format	`--replicate-do-table=name`	
Permitted Values	**Type**	string

Creates a replication filter by telling the slave SQL thread to restrict replication to a given 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`. See Section 16.2.5, “How Servers Evaluate Replication Filtering Rules”.

In MySQL 5.7.3 and later, you can also create such a filter by issuing a `CHANGE REPLICATION FILTER REPLICATE_DO_TABLE` statement.

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`***

Command-Line Format	`--replicate-ignore-table=name`	
Permitted Values	**Type**	string

Creates a replication filter by telling 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`. See Section 16.2.5, “How Servers Evaluate Replication Filtering Rules”.

In MySQL 5.7.3 and later, you can also create such a filter by issuing a `CHANGE REPLICATION FILTER REPLICATE_IGNORE_TABLE` statement.

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`***

Command-Line Format	`--replicate-rewrite-db=old_name->new_name`	
Permitted Values	**Type**	string

Tells the slave to create a replication filter that translates 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. 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.

In MySQL 5.7.3 and later, you can also create such a filter by issuing a `CHANGE REPLICATION FILTER REPLICATE_REWRITE_DB` statement.

Statements in which table names are qualified with database names when using this option do not work with table-level replication filtering options such as `--replicate-do-table`. Suppose we have a database named `a` on the master, one named `b` on the slave, each containing a table `t`, and have started the master with `--replicate-rewrite-db='a->b'`. At a later point in time, we execute `DELETE FROM a.t`. In this case, no relevant filtering rule works, for the reasons shown here:

a. `--replicate-do-table=a.t` does not work because the slave has table `t` in database `b`.

b. `--replicate-do-table=b.t` does not match the original statement and so is ignored.

c. `--replicate-do-table=*.t` is handled identically to `--replicate-do-table=a.t`, and thus does not work, either.

Similarly, the `--replication-rewrite-db` option does not work with cross-database updates.

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`

Command-Line Format	`--replicate-same-server-id`	
Permitted Values	**Type**	boolean
	Default	`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` 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`, 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`***

Command-Line Format	`--replicate-wild-do-table=name`	
Permitted Values	**Type**	string

Creates a replication filter by telling 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 Section 16.2.5, “How Servers Evaluate Replication Filtering Rules”.

In MySQL 5.7.3 and later, you can also create such a filter by issuing a `CHANGE REPLICATION FILTER REPLICATE_WILD_DO_TABLE` statement.

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

Example: `--replicate-wild-do-table=foo%.bar%` 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%.%`, 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`. 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`.

- `--replicate-wild-ignore-table=`***`db_name.tbl_name`***

Command-Line Format	`--replicate-wild-ignore-table=name`	
Permitted Values	**Type**	string

Creates a replication filter which keeps the slave thread from replicating a statement in which 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 Section 16.2.5, “How Servers Evaluate Replication Filtering Rules”.

In MySQL 5.7.3 and later, you can also create such a filter by issuing a `CHANGE REPLICATION FILTER REPLICATE_WILD_IGNORE_TABLE` statement.

Example: `--replicate-wild-ignore-table=foo%.bar%` 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` option. The rules for including literal wildcard characters in the option value are the same as for `--replicate-wild-ignore-table` as well.

- `--report-host=`***`host_name`***

Command-Line Format	`--report-host=host_name`

System Variable	**Name**	`report_host`
	Variable Scope	Global
	Dynamic Variable	No
Permitted Values	**Type**	string

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**
>
> 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`***

Command-Line Format	`--report-password=name`	
System Variable	**Name**	`report_password`
	Variable Scope	Global
	Dynamic Variable	No
Permitted Values	**Type**	string

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` 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`***

Command-Line Format	`--report-port=#`	
System Variable	**Name**	`report_port`
	Variable Scope	Global
	Dynamic Variable	No
Permitted Values	**Type**	integer
	Default	`[slave_port]`
	Min Value	`0`
	Max Value	`65535`

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.

The default value for this option is the port number actually used by the slave (Bug #13333431). This is also the default value displayed by `SHOW SLAVE HOSTS`.

- `--report-user=`***`user_name`***

Command-Line Format	`--report-user=name`	
System Variable	Name	`report_user`
	Variable Scope	Global
	Dynamic Variable	No
Permitted Values	Type	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` 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`

Command-Line Format	`--show-slave-auth-info`	
Permitted Values	Type	boolean
	Default	`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` and `--report-password` options.

- `--slave-checkpoint-group=`***`#`***

Command-Line Format	`--slave-checkpoint-group=#`	
Permitted Values	Type	integer
	Default	`512`
	Min Value	`32`
	Max Value	`524280`
	Block Size	`8`

Sets the maximum number of transactions that can be processed by a multi-threaded slave before a checkpoint operation is called to update its status as shown by `SHOW SLAVE STATUS`. Setting this option has no effect on slaves for which multi-threading is not enabled.

> **Note**

> Multi-threaded slaves are not currently supported by NDB Cluster, which silently ignores the setting for this option. See Section 21.6.3, “Known Issues in NDB Cluster Replication”, for more information.

This option works in combination with the `--slave-checkpoint-period` option in such a way that, when either limit is exceeded, the checkpoint is executed and the counters tracking both the number of transactions and the time elapsed since the last checkpoint are reset.

The minimum allowed value for this option is 32, unless the server was built using `-DWITH_DEBUG`, in which case the minimum value is 1. The effective value is always a multiple of 8; you can set it to a value that is not such a multiple, but the server rounds it down to the next lower multiple of 8 before storing the value. (*Exception*: No such rounding is performed by the debug server.) Regardless of how the server was built, the default value is 512, and the maximum allowed value is 524280.

- `--slave-checkpoint-period=#`

Command-Line Format	`--slave-checkpoint-period=#`	
Permitted Values	**Type**	integer
	Default	`300`
	Min Value	`1`
	Max Value	`4G`

Sets the maximum time (in milliseconds) that is allowed to pass before a checkpoint operation is called to update the status of a multi-threaded slave as shown by `SHOW SLAVE STATUS`. Setting this option has no effect on slaves for which multi-threading is not enabled.

> **Note**
>
> Multi-threaded slaves are not currently supported by NDB Cluster, which silently ignores the setting for this option. See Section 21.6.3, “Known Issues in NDB Cluster Replication”, for more information.

This option works in combination with the `--slave-checkpoint-group` option in such a way that, when either limit is exceeded, the checkpoint is executed and the counters tracking both the number of transactions and the time elapsed since the last checkpoint are reset.

The minimum allowed value for this option is 1, unless the server was built using `-DWITH_DEBUG`, in which case the minimum value is 0. Regardless of how the server was built, the default value is 300, and the maximum possible value is 4294967296 (4GB).

- `--slave-parallel-workers`

Command-Line Format	`--slave-parallel-workers=#`	
Permitted Values	**Type**	integer

	Default	`0`
	Min Value	`0`
	Max Value	`1024`

Sets the number of slave applier threads for executing replication transactions in parallel. Setting this variable to a number greater than 0 creates a multi-threaded slave with this number of applier threads. When set to 0 (the default) parallel execution is disabled and the slave uses a single applier thread.

A multi-threaded slave provides parallel execution by using a coordinator thread and the number of applier threads configured by this option. The way which transactions are distributed among applier threads is configured by `--slave-parallel-type`. For more information about multi-threaded slaves see `slave-parallel-workers`.

> **Note**
>
> Multi-threaded slaves are not currently supported by NDB Cluster, which silently ignores the setting for this option. See Section 21.6.3, “Known Issues in NDB Cluster Replication”, for more information.

- `--slave-pending-jobs-size-max=#`

Command-Line Format	`--slave-pending-jobs-size-max=#`	
Permitted Values	**Type**	integer
	Default	`16M`
	Min Value	`1024`
	Max Value	`18EB`
	Block Size	`1024`

For multi-threaded slaves, this option sets the maximum amount of memory (in bytes) available to slave worker queues holding events not yet applied. Setting this option has no effect on slaves for which multi-threading is not enabled.

The minimum possible value for this option is 1024; the default is 16MB. The maximum possible value is 18446744073709551615 (16 exabytes). Values that are not exact multiples of 1024 are rounded down to the next-highest multiple of 1024 prior to being stored.

> **Important**
>
> The value for this option must not be less than the master's value for `max_allowed_packet`; otherwise a slave worker queue may become full while there remain events coming from the master to be processed.

- `--skip-slave-start`

Command-Line Format	`--skip-slave-start`	
Permitted Values	**Type**	boolean
	Default	`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}`

Command-Line Format	`--slave-compressed-protocol`	
System Variable	**Name**	`slave_compressed_protocol`
	Variable Scope	Global
	Dynamic Variable	Yes
Permitted Values	**Type**	boolean
	Default	`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=`***`dir_name`***

Command-Line Format	`--slave-load-tmpdir=dir_name`	
System Variable	**Name**	`slave_load_tmpdir`
	Variable Scope	Global
	Dynamic Variable	No
Permitted Values	**Type**	directory name
	Default	`/tmp`

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` 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-max-allowed-packet=`***`bytes`***

Command-Line Format	`--slave-max-allowed-packet=#`	
Permitted Values	**Type**	integer
	Default	`1073741824`
	Min Value	`1024`
	Max Value	`1073741824`

This option sets the maximum packet size in bytes for the slave SQL and I/O threads, so that large updates using row-based replication do not cause replication to fail because an update exceeded `max_allowed_packet`. (Bug #12400221, Bug #60926)

The corresponding server variable `slave_max_allowed_packet` always has a value that is a positive integer multiple of 1024; if you set it to some value that is not such a multiple, the value is automatically rounded down to the next highest multiple of 1024. (For example, if you start the server with `--slave-max-allowed-packet=10000`, the value used is 9216; setting 0 as the value causes 1024 to be used.) A truncation warning is issued in such cases.

The maximum (and default) value is 1073741824 (1 GB); the minimum is 1024.

- `--slave-net-timeout=`***`seconds`***

Command-Line Format	`--slave-net-timeout=#`	
System Variable	**Name**	`slave_net_timeout`
	Variable Scope	Global
	Dynamic Variable	Yes
Permitted Values	**Type**	integer
	Default	`3600`
	Min Value	`1`
Permitted Values (>= 5.7.7)	**Type**	integer
	Default	`60`
	Min Value	`1`

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 `MASTER_CONNECT_RETRY` option for the `CHANGE MASTER TO` statement, and the number of reconnection attempts is limited by the `--master-retry-count` option. Prior to MySQL 5.7.7, the default was 3600 seconds (one hour). In MySQL 5.7.7 and later the default is 60 (one minute).

- `--slave-parallel-type=`***`type`***

Introduced	5.7.2	
Command-Line Format	`--slave-parallel-type=type`	
Permitted Values	**Type**	enumeration
	Default	`DATABASE`
	Valid Values	`DATABASE`
		`LOGICAL_CLOCK`

When using a multi-threaded slave (`slave_parallel_workers` is greater than 0), this option specifies the policy used to decide which transactions are allowed to execute in parallel on the slave. The possible values are:

- `DATABASE`: Transactions that update different databases are applied in parallel. This value is only appropriate if data is partitioned into multiple databases which are being updated independently and concurrently on the master. Only recommended if there are no cross-database constraints, as such constraints may be violated on the slave.

- `LOGICAL_CLOCK`: Transactions that are part of the same binary log group commit on a master are applied in parallel on a slave. There are no cross-database constraints, and data does not need to be partitioned into multiple databases.

Regardless of the value of this variable, there is no special configuration required on the master. When `slave_preserve_commit_order=1`, you can only use `LOGICAL_CLOCK`. If your replication topology uses multiple levels of slaves, `LOGICAL_CLOCK` may achieve less parallelization for each level the slave is away from the master.

- `slave-rows-search-algorithms=`***`list`***

Command-Line Format	`--slave-rows-search-algorithms=list`	
Permitted Values	**Type**	set
	Default	`TABLE_SCAN,INDEX_SCAN`
	Valid Values	`TABLE_SCAN,INDEX_SCAN`
		`INDEX_SCAN,HASH_SCAN`
		`TABLE_SCAN,HASH_SCAN`
		`TABLE_SCAN,INDEX_SCAN,HASH_SCAN` (equivalent to INDEX_SCAN,HASH_SCAN)

When preparing batches of rows for row-based logging and replication, this option controls how the rows are searched for matches—that is, whether or not hashing is used for searches using a primary or unique key, some other key, or no key at all. This option takes a comma-separated list of any 2 (or possibly 3) values from the list `INDEX_SCAN`, `TABLE_SCAN`, `HASH_SCAN`. The list need not be quoted, but must contain no spaces, whether or not quotes are used. Possible combinations (lists) and their effects are shown in the following table:

Index used / option value	`INDEX_SCAN,HASH_SCAN` **or** `INDEX_SCAN,TABLE_SCAN,HASH_SCAN`	`INDEX_SCAN,TABLE_SCAN`	`TABLE_SCAN,HASH_SCAN`
Primary key or unique key	Index scan	Index scan	Hash scan over index

Index used / option value	`INDEX_SCAN,HASH_SCAN` or `INDEX_SCAN,TABLE_SCAN,HASH_SCAN`	`INDEX_SCAN,TABLE_SCAN`	`TABLE_SCAN,HASH_SCAN`
(Other) Key	Hash scan over index	Index scan	Hash scan over index
No index	Hash scan	Table scan	Hash scan

The order in which the algorithms are specified in the list does not make any difference in the order in which they are displayed by a `SELECT` or `SHOW VARIABLES` statement (which is the same as that used in the table just shown previously).The default value is `TABLE_SCAN,INDEX_SCAN`, which means that all searches that can use indexes do use them, and searches without any indexes use table scans.

Specifying `INDEX_SCAN,TABLE_SCAN,HASH_SCAN` has the same effect as specifying `INDEX_SCAN,HASH_SCAN`. To use hashing for any searches that does not use a primary or unique key, set this option to `INDEX_SCAN,HASH_SCAN`. To force hashing for *all* searches, set it to `TABLE_SCAN,HASH_SCAN`.

> ### Note
>
> There is only a performance advantage for `INDEX_SCAN` and `HASH_SCAN` if the row events are big enough. The size of row events is configured using `--binlog-row-event-max-size`. For example, suppose a `DELETE` statement which deletes 25,000 rows generates large `Delete_row_event` events. In this case if `slave_rows_search_algorithms` is set to `INDEX_SCAN` or `HASH_SCAN` there is a performance improvement. However, if there are 25,000 `DELETE` statements and each is represented by a separate event then setting `slave_rows_search_algorithms` to `INDEX_SCAN` or `HASH_SCAN` provides no performance improvement while executing these separate events.

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

Command-Line Format	`--slave-skip-errors=name`	
System Variable	**Name**	`slave_skip_errors`
	Variable Scope	Global
	Dynamic Variable	No
Permitted Values	**Type**	string
	Default	`OFF`
	Valid Values	`OFF`
		`[list of error codes]`
		`all`

		`ddl_exist_errors`
Permitted Values	**Type**	string
	Default	`OFF`
	Valid Values	`OFF`
		`[list of error codes]`
		`all`
		`ddl_exist_errors`
Permitted Values	**Type**	string
	Default	`OFF`
	Valid Values	`OFF`
		`[list of error codes]`
		`all`
		`ddl_exist_errors`

Normally, replication stops when an error occurs on the slave, which gives you the opportunity to resolve the inconsistency in the data manually. This option causes 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`. Appendix B, *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*.

MySQL 5.7 supports an additional shorthand value `ddl_exist_errors`, which is equivalent to the error code list `1007,1008,1050,1051,1054,1060,1061,1068,1094,1146`.

Examples:

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

- `--slave-sql-verify-checksum={0|1}`

Command-Line Format	`--slave-sql-verify-checksum=value`	
Permitted Values	**Type**	boolean

	Default	`0`
	Valid Values	`0`
		`1`

When this option is enabled, the slave examines checksums read from the relay log, in the event of a mismatch, the slave stops with an error. Disabled by default.

The following options are used internally by the MySQL test suite for replication testing and debugging. They are not intended for use in a production setting.

- `--abort-slave-event-count`

Command-Line Format	`--abort-slave-event-count=#`	
Permitted Values	**Type**	integer
	Default	`0`
	Min Value	`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.

- `--disconnect-slave-event-count`

Command-Line Format	`--disconnect-slave-event-count=#`	
Permitted Values	**Type**	integer
	Default	`0`

Options for Logging Slave Status to Tables

MySQL 5.7 supports logging of replication slave status information to tables rather than files. Writing of the master info log and the relay log info log can be configured separately using the two server options listed here:

- `--master-info-repository={FILE|TABLE}`

Command-Line Format	`--master-info-repository=FILE|TABLE`	
Permitted Values	**Type**	string
	Default	`FILE`
	Valid Values	`FILE`
		`TABLE`

This option causes the server to write its master info log to a file or a table. The name of the file defaults to `master.info`; you can change the name of the file using the `--master-info-file` server option.

The default value for this option is `FILE`. If you use `TABLE`, the log is written to the `slave_master_info` table in the `mysql` database.

- `--relay-log-info-repository={FILE|TABLE}`

Command-Line Format	`--relay-log-info-repository=FILE|TABLE`	
Permitted Values	**Type**	string
	Default	`FILE`
	Valid Values	`FILE`
		`TABLE`

This option causes the server to log its relay log info to a file or a table. The name of the file defaults to `relay-log.info`; you can change the name of the file using the `--relay-log-info-file` server option.

The default value for this option is `FILE`. If you use `TABLE`, the log is written to the `slave_relay_log_info` table in the `mysql` database.

These options can be used to make replication slaves resilient to unexpected halts. See Section 16.3.2, “Handling an Unexpected Halt of a Replication Slave”, for more information.

The info log tables and their contents are considered local to a given MySQL Server. They are not replicated, and changes to them are not written to the binary log.

For more information, see Section 16.2.4, “Replication Relay and Status Logs”.

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`

Command-Line Format	`--init-slave=name`	
System Variable	**Name**	`init_slave`
	Variable Scope	Global
	Dynamic Variable	Yes
Permitted Values	**Type**	string

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. The setting of this variable takes effect for subsequent `START SLAVE` statements.

Note

> The SQL thread sends an acknowledgment to the client before it executes `init_slave`. Therefore, it is not guaranteed that `init_slave` has been executed when `START SLAVE` returns. See Section 13.4.2.6, "START SLAVE Syntax", for more information.

- `log_slow_slave_statements`

Introduced	5.7.1	
System Variable	**Name**	`log_slow_slave_statements`
	Variable Scope	Global
	Dynamic Variable	Yes
Permitted Values	**Type**	boolean
	Default	`OFF`

When the slow query log is enabled, this variable enables logging for queries that have taken more than `long_query_time` seconds to execute on the slave. This variable was added in MySQL 5.7.1. Setting this variable has no immediate effect. The state of the variable applies on all subsequent `START SLAVE` statements.

- `master_info_repository`

Command-Line Format	`--master-info-repository=FILE|TABLE`	
System Variable	**Name**	`master_info_repository`
	Variable Scope	Global
	Dynamic Variable	Yes
Permitted Values	**Type**	string
	Default	`FILE`
	Valid Values	`FILE`
		`TABLE`

The setting of this variable determines whether the slave logs master status and connection information to a `FILE` (`master.info`), or to a `TABLE` (`mysql.slave_master_info`). You can only change the value of this variable when no replication threads are executing.

The setting of this variable also has a direct influence on the effect had by the setting of the `sync_master_info` system variable; see that variable's description for further information.

This variable must be set to `TABLE` before configuring multiple replication channels. If you are using multiple replication channels then you cannot set this variable back to `FILE`.

- `relay_log`

Command-Line Format	`--relay-log=file_name`	
System Variable	**Name**	`relay_log`
	Variable Scope	Global

	Dynamic Variable	No
Permitted Values	**Type**	file name

The base name of the relay log file, with no paths and no file extension. By default `relay-log`. The file name of individual files for the default replication channel is `relay-log.XXXXXX`, and for additional replication channels is `relay-log-`***`channel`***`.XXXXXX`.

- `relay_log_basename`

System Variable	**Name**	`relay_log_basename`
	Variable Scope	Global
	Dynamic Variable	No
Permitted Values	**Type**	file name
	Default	`datadir + '/' + hostname + '-relay-bin'`

Holds the name and complete path to the relay log file.

- `relay_log_index`

Command-Line Format	`--relay-log-index`	
System Variable	**Name**	`relay_log_index`
	Variable Scope	Global
	Dynamic Variable	No
Permitted Values	**Type**	file name
	Default	`*host_name*-relay-bin.index`

The name of the relay log index file for the default replication channel. 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`

Command-Line Format	`--relay-log-info-file=file_name`	
System Variable	**Name**	`relay_log_info_file`
	Variable Scope	Global
	Dynamic Variable	No
Permitted Values	**Type**	file name
	Default	`relay-log.info`

The name of the file in which the slave records information about the relay logs, when `relay_log_info_repository=FILE`. If `relay_log_info_repository=TABLE`, it is the file name that would be used in case the repository was changed to `FILE`). The default name is `relay-log.info` in the data directory.

- `relay_log_info_repository`

System Variable	**Name**	`relay_log_info_repository`

	Variable Scope	Global
	Dynamic Variable	Yes
Permitted Values	**Type**	string
	Default	`FILE`
	Valid Values	`FILE`
		`TABLE`

This variable determines whether the slave's position in the relay logs is written to a `FILE` (`relay-log.info`) or to a `TABLE` (`mysql.slave_relay_log_info`). You can only change the value of this variable when no replication threads are executing.

The setting of this variable also has a direct influence on the effect had by the setting of the `sync_relay_log_info` system variable; see that variable's description for further information.

This variable must be set to `TABLE` before configuring multiple replication channels. If you are using multiple replication channels then you cannot set this variable back to `FILE`.

- `relay_log_recovery`

Command-Line Format	`--relay-log-recovery`	
System Variable	**Name**	`relay_log_recovery`
	Variable Scope	Global
	Dynamic Variable	No
Permitted Values	**Type**	boolean
	Default	`FALSE`

Enables automatic relay log recovery immediately following server startup. The recovery process creates a new relay log file, initializes the SQL thread position to this new relay log, and initializes the I/O thread to the SQL thread position. Reading of the relay log from the master then continues. In MySQL 5.7, this global variable is read-only; its value can be changed by starting the slave with the `--relay-log-recovery` option, which should be used following an unexpected halt of a replication slave to ensure that no possibly corrupted relay logs are processed. See Section 16.3.2, “Handling an Unexpected Halt of a Replication Slave” for more information.

This variable also interacts with `relay-log-purge`, which controls purging of logs when they are no longer needed. Enabling the `--relay-log-recovery` option when `relay-log-purge` is disabled risks reading the relay log from files that were not purged, leading to data inconsistency.

When `relay_log_recovery` is enabled and the slave has stopped due to errors encountered while running in multi-threaded mode, you can use `START SLAVE UNTIL SQL_AFTER_MTS_GAPS` to ensure that all gaps are processed before switching back to single-threaded mode or executing a `CHANGE MASTER TO` statement.

- `rpl_stop_slave_timeout`

Introduced	5.7.2

Command-Line Format	--rpl-stop-slave-timeout=seconds	
System Variable	Name	rpl_stop_slave_timeout
	Variable Scope	Global
	Dynamic Variable	Yes
Permitted Values	Type	integer
	Default	31536000
	Min Value	2
	Max Value	31536000

In MySQL 5.7.2 and later, you can control the length of time (in seconds) that `STOP SLAVE` waits before timing out by setting this variable. This can be used to avoid deadlocks between `STOP SLAVE` and other slave SQL statements using different client connections to the slave. The maximum and default value of `rpl_stop_slave_timeout` is 31536000 seconds (1 year). The minimum is 2 seconds. Changes to this variable take effect for subsequent `STOP SLAVE` statements. This variable affects only the client that issues a `STOP SLAVE` statement. When the timeout is reached, the issuing client stops waiting for the slave threads to stop, but the slave threads continue to try to stop.

- `slave_checkpoint_group`

Command-Line Format	--slave-checkpoint-group=#	
System Variable	Name	slave_checkpoint_group=#
	Variable Scope	Global
	Dynamic Variable	Yes
Permitted Values	Type	integer
	Default	512
	Min Value	32
	Max Value	524280
	Block Size	8

Sets the maximum number of transactions that can be processed by a multi-threaded slave before a checkpoint operation is called to update its status as shown by `SHOW SLAVE STATUS`. Setting this variable has no effect on slaves for which multi-threading is not enabled. Setting this variable has no immediate effect. The state of the variable applies on all subsequent `START SLAVE` commands.

> ### Note
>
> Multi-threaded slaves are not currently supported by NDB Cluster, which silently ignores the setting for this variable. See Section 21.6.3, “Known Issues in NDB Cluster Replication”, for more information.

This variable works in combination with the `slave_checkpoint_period` system variable in such a way that, when either limit is exceeded, the checkpoint is executed and the counters tracking both the number of transactions and the time elapsed since the last checkpoint are reset.

The minimum allowed value for this variable is 32, unless the server was built using `-DWITH_DEBUG`, in which case the minimum value is 1. The effective value is always a multiple of 8; you can set it to a value that is not such a multiple, but the server rounds it down to the next lower multiple of 8 before storing the value. (*Exception*: No such rounding is performed by the debug server.) Regardless of how the server was built, the default value is 512, and the maximum allowed value is 524280.

- `slave_checkpoint_period`

Command-Line Format	`--slave-checkpoint-period=#`	
System Variable	**Name**	`slave_checkpoint_period=#`
	Variable Scope	Global
	Dynamic Variable	Yes
Permitted Values	**Type**	integer
	Default	`300`
	Min Value	`1`
	Max Value	`4G`

Sets the maximum time (in milliseconds) that is allowed to pass before a checkpoint operation is called to update the status of a multi-threaded slave as shown by `SHOW SLAVE STATUS`. Setting this variable has no effect on slaves for which multi-threading is not enabled. Setting this variable takes effect for all replication channels immediately, including running channels.

> **Note**
>
> Multi-threaded slaves are not currently supported by NDB Cluster, which silently ignores the setting for this variable. See Section 21.6.3, “Known Issues in NDB Cluster Replication”, for more information.

This variable works in combination with the `slave_checkpoint_group` system variable in such a way that, when either limit is exceeded, the checkpoint is executed and the counters tracking both the number of transactions and the time elapsed since the last checkpoint are reset.

The minimum allowed value for this variable is 1, unless the server was built using `-DWITH_DEBUG`, in which case the minimum value is 0. Regardless of how the server was built, the default value is 300, and the maximum possible value is 4294967296 (4GB).

- `slave_compressed_protocol`

Command-Line Format	`--slave-compressed-protocol`	
System Variable	**Name**	`slave_compressed_protocol`
	Variable Scope	Global
	Dynamic Variable	Yes
Permitted Values	**Type**	boolean
	Default	`OFF`

Whether to use compression of the slave/master protocol if both the slave and the master support it. Changes to this variable take effect on subsequent connection attempts; this includes after issuing a `START SLAVE` statement, as well as reconnections made by a running I/O thread (for example after issuing a `CHANGE MASTER TO MASTER_RETRY_COUNT` statement).

- `slave_exec_mode`

Command-Line Format	--slave-exec-mode=mode	
System Variable	Name	slave_exec_mode
	Variable Scope	Global
	Dynamic Variable	Yes
Permitted Values	Type	enumeration
	Default	STRICT (ALL)
	Default	IDEMPOTENT (NDB)
	Valid Values	IDEMPOTENT
		STRICT

Controls how a slave thread resolves conflicts and errors during replication. `STRICT` mode is the default, and is suitable for most cases. `IDEMPOTENT` mode causes suppression of duplicate-key and no-key-found errors. This mode should only be used with NDB Cluster Replication in some special scenarios, such as multi-master replication, and circular replication. (See Section 21.6.10, “NDB Cluster Replication: Multi-Master and Circular Replication”, and Section 21.6.11, “NDB Cluster Replication Conflict Resolution”, for more information.) Setting this variable takes effect for all replication channels immediately, including running channels.

> ### Warning
>
> The **mysqld** supplied with NDB Cluster ignores any value explicitly set for `slave_exec_mode`, and always treats it as `IDEMPOTENT`.

`IDEMPOTENT` mode is supported only by `NDB` and is used when replicating `NDB` to `InnoDB`.

- `slave_load_tmpdir`

Command-Line Format	--slave-load-tmpdir=dir_name	
System Variable	Name	slave_load_tmpdir
	Variable Scope	Global
	Dynamic Variable	No
Permitted Values	Type	directory name
	Default	/tmp

The name of the directory where the slave creates temporary files for replicating `LOAD DATA INFILE` statements. Setting this variable takes effect for all replication channels immediately, including running channels.

- slave_max_allowed_packet

System Variable	Name	slave_max_allowed_packet
	Variable Scope	Global
	Dynamic Variable	Yes
Permitted Values	Type	integer
	Default	1073741824
	Min Value	1024
	Max Value	1073741824

  This variable sets the maximum packet size for the slave SQL and I/O threads, so that large updates using row-based replication do not cause replication to fail because an update exceeded max_allowed_packet. Setting this variable takes effect for all replication channels immediately, including running channels.

  This global variable always has a value that is a positive integer multiple of 1024; if you set it to some value that is not, the value is rounded down to the next highest multiple of 1024 for it is stored or used; setting slave_max_allowed_packet to 0 causes 1024 to be used. (A truncation warning is issued in all such cases.) The default and maximum value is 1073741824 (1 GB); the minimum is 1024.

  slave_max_allowed_packet can also be set at startup, using the --slave-max-allowed-packet option.

- slave_net_timeout

Command-Line Format	--slave-net-timeout=#	
System Variable	Name	slave_net_timeout
	Variable Scope	Global
	Dynamic Variable	Yes
Permitted Values	Type	integer
	Default	3600
	Min Value	1
Permitted Values (>= 5.7.7)	Type	integer
	Default	60
	Min Value	1

  The number of seconds to wait for more data from a master/slave connection before aborting the read. Setting this variable has no immediate effect. The state of the variable applies on all subsequent START SLAVE commands.

- slave_parallel_type=***type***

Introduced	5.7.2	
Command-Line Format	--slave-parallel-type=type	
Permitted Values	Type	enumeration

	Default	DATABASE
	Valid Values	DATABASE
		LOGICAL_CLOCK

When using a multi-threaded slave (slave_parallel_workers is greater than 0), this variable specifies the policy used to decide which transactions are allowed to execute in parallel on the slave. See --slave-parallel-type for more information.

- slave_parallel_workers

Command-Line Format	--slave-parallel-workers=#	
System Variable	**Name**	slave_parallel_workers
	Variable Scope	Global
	Dynamic Variable	Yes
Permitted Values	**Type**	integer
	Default	0
	Min Value	0
	Max Value	1024

Sets the number of slave applier threads for executing replication transactions in parallel. Setting this variable to a number greater than 0 creates a multi-threaded slave with this number of applier threads. When set to 0 (the default) parallel execution is disabled and the slave uses a single applier thread. Setting slave_parallel_workers has no immediate effect. The state of the variable applies on all subsequent START SLAVE statements.

> **Note**
>
> Multi-threaded slaves are not currently supported by NDB Cluster, which silently ignores the setting for this variable. See Section 21.6.3, "Known Issues in NDB Cluster Replication", for more information.

A multi-threaded slave provides parallel execution by using a coordinator thread and the number of applier threads configured by this variable. The way which transactions are distributed among applier threads is configured by slave_parallel_type. The transactions that the slave applies in parallel may commit out of order, unless slave_preserve_commit_order=1. Therefore, checking for the most recently executed transaction does not guarantee that all previous transactions from the master have been executed on the slave. This has implications for logging and recovery when using a multi-threaded slave. For example, on a multi-threaded slave the START SLAVE UNTIL statement only supports using SQL_AFTER_MTS_GAPS.

In MySQL 5.7.5 and later, retrying of transactions is supported when multi-threading is enabled on a slave. In previous versions, slave_transaction_retries was treated as equal to 0 when using multi-threaded slaves.

- `slave_pending_jobs_size_max`

System Variable	**Name**	`slave_pending_jobs_size_max`
	Variable Scope	Global
	Dynamic Variable	Yes
Permitted Values	**Type**	integer
	Default	`16M`
	Min Value	`1024`
	Max Value	`18EB`
	Block Size	`1024`

For multi-threaded slaves, this variable sets the maximum amount of memory (in bytes) available to slave worker queues holding events not yet applied. Setting this variable has no effect on slaves for which multi-threading is not enabled. Setting this variable has no immediate effect. The state of the variable applies on all subsequent `START SLAVE` commands.

The minimum possible value for this variable is 1024; the default is 16MB. The maximum possible value is 18446744073709551615 (16 exabytes). Values that are not exact multiples of 1024 are rounded down to the next-highest multiple of 1024 prior to being stored.

> **Important**
>
> The value of this variable must not be less than the master's value for `max_allowed_packet`; otherwise a slave worker queue may become full while there remain events coming from the master to be processed.

- `slave_preserve_commit_order`

Introduced	5.7.5	
Command-Line Format	`--slave-preserve-commit-order=value`	
System Variable	**Name**	`slave_preserve_commit_order`
	Variable Scope	Global
	Dynamic Variable	Yes
Permitted Values	**Type**	boolean
	Default	`0`
	Valid Values	`0`
		`1`

For multi-threaded slaves, enabling this variable ensures that transactions are externalized on the slave in the same order as they appear in the slave's relay log. Setting this variable has no effect on slaves for which multi-threading is not enabled. All replication threads (for all replication channels if you are using multiple replication channels) must be stopped before changing this variable. `--log-bin` and `--log-slave-updates` must be enabled on the slave. In addition `--slave-parallel-type` must be set to `LOGICAL_CLOCK`.

Once a multi-threaded slave has been started, transactions can begin to execute in parallel. With `slave_preserve_commit_order` enabled, the executing thread waits until all previous transactions are committed before committing. While the slave thread is waiting for other workers to commit their transactions it reports its status as `Waiting for preceding transaction to commit`. (Prior to MySQL 5.7.8, this was shown as `Waiting for its turn to commit`.) Enabling this mode on a multi-threaded slave ensures that it never enters a state that the master was not in. This makes it well suited to using replication for read scale-out. See Section 16.3.4, “Using Replication for Scale-Out”.

When using a multi-threaded slave, if `slave_preserve_commit_order` is not enabled, there is a chance of gaps in the sequence of transactions that have been executed from the slave's relay log. When this option is enabled, there is not this chance of gaps, but `Exec_master_log_pos` may be behind the position up to which has been executed. See Section 16.4.1.34, “Replication and Transaction Inconsistencies” for more information.

- `slave_rows_search_algorithms`

System Variable	**Name**	`slave_rows_search_algorithms=list`
	Variable Scope	Global
	Dynamic Variable	Yes
Permitted Values	**Type**	set
	Default	`TABLE_SCAN,INDEX_SCAN`
	Valid Values	`TABLE_SCAN,INDEX_SCAN`
		`INDEX_SCAN,HASH_SCAN`
		`TABLE_SCAN,HASH_SCAN`
		`TABLE_SCAN,INDEX_SCAN,HASH_SCAN` (equivalent to INDEX_SCAN,HASH_SCAN)

When preparing batches of rows for row-based logging and replication, this variable controls how the rows are searched for matches—that is, whether or not hashing is used for searches using a primary or unique key, using some other key, or using no key at all. Setting this variable takes effect for all replication channels immediately, including running channels.

This variable takes a comma-separated list of at least 2 values from the list `INDEX_SCAN`, `TABLE_SCAN`, `HASH_SCAN`. The value expected as a string, so the value must be quoted. In addition, the value must not contain any spaces. Possible combinations (lists) and their effects are shown in the following table:

Index used / option value	`INDEX_SCAN,HASH_SCAN` **or** `INDEX_SCAN,TABLE_SCAN,HASH_SCAN`	`INDEX_SCAN,TABLE_SCAN`	`TABLE_SCAN,HASH_SCAN`

Primary key or unique key	Index scan	index scan	Index hash
(Other) Key	Index hash	Index scan	Index hash
No index	Table hash	Table scan	Table hash

The order in which the algorithms are specified in the list does not make any difference in the order in which they are displayed by a `SELECT` or `SHOW VARIABLES` statement, as shown here:

```
mysql> SET GLOBAL slave_rows_search_algorithms = "INDEX_SCAN,TABLE_SCAN";
Query OK, 0 rows affected (0.00 sec)

mysql> SHOW VARIABLES LIKE '%algorithms%';
+------------------------------+-----------------------+
| Variable_name                | Value                 |
+------------------------------+-----------------------+
| slave_rows_search_algorithms | TABLE_SCAN,INDEX_SCAN |
+------------------------------+-----------------------+
1 row in set (0.00 sec)

mysql> SET GLOBAL slave_rows_search_algorithms = "TABLE_SCAN,INDEX_SCAN";
Query OK, 0 rows affected (0.00 sec)

mysql> SHOW VARIABLES LIKE '%algorithms%';
+------------------------------+-----------------------+
| Variable_name                | Value                 |
+------------------------------+-----------------------+
| slave_rows_search_algorithms | TABLE_SCAN,INDEX_SCAN |
+------------------------------+-----------------------+
1 row in set (0.00 sec)
```

The default value is `TABLE_SCAN,INDEX_SCAN`, which means that all searches that can use indexes do use them, and searches without any indexes use table scans.

Specifying `INDEX_SCAN,TABLE_SCAN,HASH_SCAN` has the same effect as specifying `INDEX_SCAN,HASH_SCAN`. To use hashing for any searches that does not use a primary or unique key, set this variable to `INDEX_SCAN,HASH_SCAN`. To force hashing for *all* searches, set it to `TABLE_SCAN,HASH_SCAN`.

- `slave_skip_errors`

Command-Line Format	`--slave-skip-errors=name`	
System Variable	**Name**	`slave_skip_errors`
	Variable Scope	Global
	Dynamic Variable	No

Permitted Values	**Type**	string
	Default	`OFF`
	Valid Values	`OFF`
		`[list of error codes]`
		`all`
		`ddl_exist_errors`
Permitted Values	**Type**	string
	Default	`OFF`
	Valid Values	`OFF`
		`[list of error codes]`
		`all`
		`ddl_exist_errors`
Permitted Values	**Type**	string
	Default	`OFF`
	Valid Values	`OFF`
		`[list of error codes]`
		`all`
		`ddl_exist_errors`

Normally, replication stops when an error occurs on the slave, which gives you the opportunity to resolve the inconsistency in the data manually. This variable causes the slave SQL thread to continue replication when a statement returns any of the errors listed in the variable value. The setting of this variable takes effect immediately, even for running replication threads.

- `slave_sql_verify_checksum`

System Variable	**Name**	`slave_sql_verify_checksum`
	Variable Scope	Global
	Dynamic Variable	Yes
Permitted Values	**Type**	boolean
	Default	`1`
	Valid Values	`0`
		`1`

Cause the slave SQL thread to verify data using the checksums read from the relay log. In the event of a mismatch, the slave stops with an error. Setting this variable takes effect for all replication channels immediately, including running channels.

> **Note**
>
> The slave I/O thread always reads checksums if possible when accepting events from over the network.

- `slave_transaction_retries`

Command-Line Format	`--slave-transaction-retries=#`	
System Variable	**Name**	`slave_transaction_retries`
	Variable Scope	Global
	Dynamic Variable	Yes
Permitted Values (32-bit platforms)	**Type**	integer
	Default	`10`
	Min Value	`0`
	Max Value	`4294967295`
Permitted Values (64-bit platforms)	**Type**	integer
	Default	`10`
	Min Value	`0`
	Max Value	`18446744073709551615`

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 `NDB`'s `TransactionDeadlockDetectionTimeout` or `TransactionInactiveTimeout`, it automatically retries `slave_transaction_retries` times before stopping with an error. The default value is 10. Setting this variable takes effect for all replication channels immediately, including running channels.

As of MySQL 5.7.5, retrying of transactions is supported when multi-threading is enabled on a slave. In previous versions, `slave_transaction_retries` was treated as equal to 0 when using multi-threaded slaves.

- `slave_type_conversions`

Command-Line Format	`--slave-type-conversions=set`	
System Variable	**Name**	`slave_type_conversions`
	Variable Scope	Global
	Dynamic Variable	No
Permitted Values (<= 5.7.1)	**Type**	set
	Default	
	Valid Values	`ALL_LOSSY`
		`ALL_NON_LOSSY`
Permitted Values (>= 5.7.2)	**Type**	set
	Default	
	Valid Values	`ALL_LOSSY`
		`ALL_NON_LOSSY`
		`ALL_SIGNED`
		`ALL_UNSIGNED`

Controls the type conversion mode in effect on the slave when using row-based replication. In MySQL 5.7.2 and later, its value is a comma-delimited set of zero or more elements from the list: `ALL_LOSSY`, `ALL_NON_LOSSY`, `ALL_SIGNED`, `ALL_UNSIGNED`. Set this variable to an empty string to disallow type conversions between the master and the slave. Setting this variable takes effect for all replication channels immediately, including running channels.

ALL_SIGNED and ALL_UNSIGNED were added in MySQL 5.7.2 (Bug#15831300). For additional information on type conversion modes applicable to attribute promotion and demotion in row-based replication, see Row-based replication: attribute promotion and demotion.

- sql_slave_skip_counter

System Variable	**Name**	sql_slave_skip_counter
	Variable Scope	Global
	Dynamic Variable	Yes
Permitted Values	**Type**	integer

The number of events from the master that a slave server should skip. Setting the option has no immediate effect. The variable applies to the next START SLAVE statement; the next START SLAVE statement also changes the value back to 0. When this variable is set to a non-zero value and there are multiple replication channels configured, the START SLAVE statement can only be used with the FOR CHANNEL ***channel*** clause.

This option is incompatible with GTID-based replication, and must not be set to a nonzero value when --gtid-mode=ON. In MySQL 5.7.1 and later, trying to do so is specifically disallowed. (Bug #15833516) If you need to skip transactions when employing GTIDs, use gtid_executed from the master instead. See Injecting empty transactions, for information about how to do this.

> **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 Section 13.4.2.5, “SET GLOBAL sql_slave_skip_counter Syntax”.

- sync_master_info

Command-Line Format	--sync-master-info=#	
System Variable	**Name**	sync_master_info
	Variable Scope	Global
	Dynamic Variable	Yes
Permitted Values (32-bit platforms)	**Type**	integer
	Default	10000
	Min Value	0
	Max Value	4294967295
Permitted Values (64-bit platforms)	**Type**	integer
	Default	10000
	Min Value	0
	Max Value	18446744073709551615

The effects of this variable on a replication slave depend on whether the slave's `master_info_repository` is set to `FILE` or `TABLE`, as explained in the following paragraphs.

master_info_repository = FILE. If the value of `sync_master_info` is greater than 0, the slave synchronizes its `master.info` file to disk (using `fdatasync()`) after every `sync_master_info` events. If it is 0, the MySQL server performs no synchronization of the `master.info` file to disk; instead, the server relies on the operating system to flush its contents periodically as with any other file.

master_info_repository = TABLE. If the value of `sync_master_info` is greater than 0, the slave updates its master info repository table after every `sync_master_info` events. If it is 0, the table is never updated.

The default value for `sync_master_info` is 10000. Setting this variable takes effect for all replication channels immediately, including running channels.

- `sync_relay_log`

Command-Line Format	`--sync-relay-log=#`	
System Variable	**Name**	`sync_relay_log`
	Variable Scope	Global
	Dynamic Variable	Yes
Permitted Values (32-bit platforms)	**Type**	integer
	Default	`10000`
	Min Value	`0`
	Max Value	`4294967295`
Permitted Values (64-bit platforms)	**Type**	integer
	Default	`10000`
	Min Value	`0`
	Max Value	`18446744073709551615`

If the value of this variable is greater than 0, the MySQL server synchronizes its relay log to disk (using `fdatasync()`) after every `sync_relay_log` events are written to the relay log. Setting this variable takes effect for all replication channels immediately, including running channels.

Setting `sync_relay_log` to 0 causes no synchronization to be done to disk; in this case, the server relies on the operating system to flush the relay log's contents from time to time as for any other file.

A value of 1 is the safest choice because in the event of a crash you lose at most one event from the relay log. However, it is also the slowest choice (unless the disk has a battery-backed cache, which makes synchronization very fast).

- `sync_relay_log_info`

Command-Line Format	`--sync-relay-log-info=#`

System Variable	**Name**	`sync_relay_log_info`
	Variable Scope	Global
	Dynamic Variable	Yes
Permitted Values (32-bit platforms)	**Type**	integer
	Default	`10000`
	Min Value	`0`
	Max Value	`4294967295`
Permitted Values (64-bit platforms)	**Type**	integer
	Default	`10000`
	Min Value	`0`
	Max Value	`18446744073709551615`

The effects of this variable on the slave depend on the server's `relay_log_info_repository` setting (`FILE` or `TABLE`), and if this is `TABLE`, additionally on whether the storage engine used by the relay log info table is transactional (such as `InnoDB`) or not (`MyISAM`). The effects of these factors on the behavior of the server for `sync_relay_log_info` values of zero and greater than zero are shown in the following table:

`sync_relay_log_info`	`relay_log_info_repository`		
	`FILE`	`TABLE`	
		Transactional	**Nontransactional**
`N` `> 0`	The slave synchronizes its `relay-log.info` file to disk (using `fdatasync()`) after every ***`N`*** transactions.	The table is updated after each transaction. (***`N`*** is effectively ignored.)	The table is updated after every ***`N`*** events.
`0`	The MySQL server performs no synchronization of the `relay-log.info` file to disk; instead, the server relies on the operating system to flush its contents periodically as with any other file.		The table is never updated.

The default value for `sync_relay_log_info` is 10000. Setting this variable takes effect for all replication channels immediately, including running channels.