



Full credit is given to the above companies including the OS that this PDF file was generated!

Rocky Enterprise Linux 9.2 Manual Pages on command 'mysqlbinlog.1'

\$ man mysqlbinlog.1

MYSQLBINLOG(1) MySQL Database System MYQSLBINLOG(1)

NAME

mysqlbinlog - utility for processing binary log files

SYNOPSIS

mysqlbinlog [options] log_file ...

DESCRIPTION

The server's binary log consists of files containing ?events? that describe modifications to database contents. The server writes these files in binary format. To display their contents in text format, use the mysqlbinlog utility. You can also use mysqlbinlog to display the contents of relay log files written by a replica server in a replication setup because relay logs have the same format as binary logs. The binary log and relay log are discussed further in Section 5.4.4, ?The Binary Log?, and Section 17.2.4, ?Relay Log and Replication Metadata Repositories?.

Invoke mysqlbinlog like this:

mysqlbinlog [options] log_file ...

For example, to display the contents of the binary log file named

binlog.000003, use this command:

```
mysqlbinlog binlog.000003
```

The output includes events contained in binlog.000003. For statement-based logging, event information includes the SQL statement, the ID of the server on which it was executed, the timestamp when the statement was executed, how much time it took, and so forth. For row-based logging, the event indicates a row change rather than an SQL statement. See Section 17.2.1, "Replication Formats", for information about logging modes.

Events are preceded by header comments that provide additional information. For example:

```
# at 141
#100309 9:28:36 server id 123  end_log_pos 245
  Query thread_id=3350  exec_time=11  error_code=0
```

In the first line, the number following at indicates the file offset, or starting position, of the event in the binary log file.

The second line starts with a date and time indicating when the statement started on the server where the event originated. For replication, this timestamp is propagated to replica servers. server id is the server_id value of the server where the event originated. end_log_pos indicates where the next event starts (that is, it is the end position of the current event + 1). thread_id indicates which thread executed the event. exec_time is the time spent executing the event, on a replication source server. On a replica, it is the difference of the end execution time on the replica minus the beginning execution time on the source. The difference serves as an indicator of how much replication lags behind the source. error_code indicates the result from executing the event. Zero means that no error occurred.

Note

When using event groups, the file offsets of events may be grouped together and the comments of events may be grouped together. Do not mistake these grouped events for blank file offsets.

The output from mysqlbinlog can be re-executed (for example, by using

it as input to mysql) to redo the statements in the log. This is useful for recovery operations after an unexpected server exit. For other usage examples, see the discussion later in this section and in Section 7.5, "Point-in-Time (Incremental) Recovery". To execute the internal-use BINLOG statements used by mysqlbinlog, the user requires the BINLOG_ADMIN privilege (or the deprecated SUPER privilege), or the REPLICATION_APPLIER privilege plus the appropriate privileges to execute each log event.

You can use mysqlbinlog to read binary log files directly and apply them to the local MySQL server. You can also read binary logs from a remote server by using the --read-from-remote-server option. To read remote binary logs, the connection parameter options can be given to indicate how to connect to the server. These options are --host, --password, --port, --protocol, --socket, and --user.

When binary log files have been encrypted, which can be done from MySQL 8.0.14 onwards, mysqlbinlog cannot read them directly, but can read them from the server using the --read-from-remote-server option. Binary log files are encrypted when the server's binlog_encryption system variable is set to ON. The SHOW BINARY LOGS statement shows whether a particular binary log file is encrypted or unencrypted. Encrypted and unencrypted binary log files can also be distinguished using the magic number at the start of the file header for encrypted log files (0xFD62696E), which differs from that used for unencrypted log files (0xFE62696E). Note that from MySQL 8.0.14, mysqlbinlog returns a suitable error if you attempt to read an encrypted binary log file directly, but older versions of mysqlbinlog do not recognise the file as a binary log file at all. For more information on binary log encryption, see Section 17.3.2, "Encrypting Binary Log Files and Relay Log Files".

When binary log transaction payloads have been compressed, which can be done from MySQL 8.0.20 onwards, mysqlbinlog versions from that release on automatically decompress and decode the transaction payloads, and print them as they would uncompressed events. Older versions of

mysqlbinlog cannot read compressed transaction payloads. When the server's `binlog_transaction_compression` system variable is set to ON, transaction payloads are compressed and then written to the server's binary log file as a single event (a `Transaction_payload_event`). With the `--verbose` option, mysqlbinlog adds comments stating the compression algorithm used, the compressed payload size that was originally received, and the resulting payload size after decompression.

Note

The end position (`end_log_pos`) that mysqlbinlog states for an individual event that was part of a compressed transaction payload is the same as the end position of the original compressed payload.

Multiple decompressed events can therefore have the same end position.

mysqlbinlog's own connection compression does less if transaction payloads are already compressed, but still operates on uncompressed transactions and headers.

For more information on binary log transaction compression, see Section 5.4.4.5, ["Binary Log Transaction Compression"](#).

When running mysqlbinlog against a large binary log, be careful that the filesystem has enough space for the resulting files. To configure the directory that mysqlbinlog uses for temporary files, use the `TMPDIR` environment variable.

mysqlbinlog sets the value of `pseudo_replica_mode` or `pseudo_slave_mode` to true before executing any SQL statements. This system variable affects the handling of XA transactions, the `original_commit_timestamp` replication delay timestamp and the `original_server_version` system variable, and unsupported SQL modes.

mysqlbinlog supports the following options, which can be specified on the command line or in the `[mysqlbinlog]` and `[client]` groups of an option file. For information about option files used by MySQL programs, see Section 4.2.2.2, ["Using Option Files"](#).

? `--help`, `-?` Display a help message and exit.

? `--base64-output=value` This option determines when events should be

displayed encoded as base-64 strings using BINLOG statements. The option has these permissible values (not case-sensitive):

? AUTO ("automatic") or UNSPEC ("unspecified") displays BINLOG statements automatically when necessary (that is, for format description events and row events). If no --base64-output option is given, the effect is the same as --base64-output=AUTO.

Note

Automatic BINLOG display is the only safe behavior if you intend to use the output of mysqlbinlog to re-execute binary log file contents. The other option values are intended only for debugging or testing purposes because they may produce output that does not include all events in executable form.

? NEVER causes BINLOG statements not to be displayed. mysqlbinlog exits with an error if a row event is found that must be displayed using BINLOG.

? DECODE-ROWS specifies to mysqlbinlog that you intend for row events to be decoded and displayed as commented SQL statements by also specifying the --verbose option. Like NEVER, DECODE-ROWS suppresses display of BINLOG statements, but unlike NEVER, it does not exit with an error if a row event is found.

For examples that show the effect of --base64-output and --verbose on row event output, see the section called ?MYSQLBINLOG ROW EVENT DISPLAY?.

? --bind-address=ip_address On a computer having multiple network interfaces, use this option to select which interface to use for connecting to the MySQL server.

? --binlog-row-event-max-size=N

??

?Command-Line Format ? --binlog-row-event-max- ?

? ? size=# ?

??

?Type ? Numeric ?

??

?Default Value ? 4294967040 ?

??

?Minimum Value ? 256 ?

??

?Maximum Value ? 18446744073709547520 ?

??

Specify the maximum size of a row-based binary log event, in bytes.

Rows are grouped into events smaller than this size if possible.

The value should be a multiple of 256. The default is 4GB.

? --character-sets-dir=dir_name The directory where character sets are installed. See Section 10.15, ?Character Set Configuration?.

? --compress Compress all information sent between the client and the server if possible. See Section 4.2.8, ?Connection Compression Control?.

This option was added in MySQL 8.0.17. As of MySQL 8.0.18 it is deprecated. Expect it to be removed in a future version of MySQL.

See the section called ?Configuring Legacy Connection Compression?.

? --compression-algorithms=value The permitted compression algorithms for connections to the server. The available algorithms are the same as for the protocol_compression_algorithms system variable. The default value is uncompressed.

For more information, see Section 4.2.8, ?Connection Compression Control?.

This option was added in MySQL 8.0.18.

? --connection-server-id=server_id --connection-server-id specifies the server ID that mysqlbinlog reports when it connects to the server. It can be used to avoid a conflict with the ID of a replica server or another mysqlbinlog process.

If the --read-from-remote-server option is specified, mysqlbinlog reports a server ID of 0, which tells the server to disconnect

after sending the last log file (nonblocking behavior). If the

--stop-never option is also specified to maintain the connection to the server, mysqlbinlog reports a server ID of 1 by default instead of 0, and --connection-server-id can be used to replace that server ID if required. See the section called "SPECIFYING THE MYSQLBINLOG SERVER ID".

? --database=db_name, -d db_name This option causes mysqlbinlog to output entries from the binary log (local log only) that occur while db_name is been selected as the default database by USE.

The --database option for mysqlbinlog is similar to the --binlog-do-db option for mysqld, but can be used to specify only one database. If --database is given multiple times, only the last instance is used.

The effects of this option depend on whether the statement-based or row-based logging format is in use, in the same way that the effects of --binlog-do-db depend on whether statement-based or row-based logging is in use.

Statement-based logging. The --database option works as follows:

- ? While db_name is the default database, statements are output whether they modify tables in db_name or a different database.
- ? Unless db_name is selected as the default database, statements are not output, even if they modify tables in db_name.
- ? There is an exception for CREATE DATABASE, ALTER DATABASE, and DROP DATABASE. The database being created, altered, or dropped is considered to be the default database when determining whether to output the statement.

Suppose that the binary log was created by executing these statements using statement-based-logging:

```
INSERT INTO test.t1 (i) VALUES(100);
INSERT INTO db2.t2 (j) VALUES(200);
USE test;
INSERT INTO test.t1 (i) VALUES(101);
INSERT INTO t1 (i) VALUES(102);
INSERT INTO db2.t2 (j) VALUES(201);
```

```
USE db2;
INSERT INTO test.t1 (i) VALUES(103);
INSERT INTO db2.t2 (j) VALUES(202);
INSERT INTO t2 (j) VALUES(203);
```

mysqlbinlog --database=test does not output the first two INSERT statements because there is no default database. It outputs the three INSERT statements following USE test, but not the three INSERT statements following USE db2.

mysqlbinlog --database=db2 does not output the first two INSERT statements because there is no default database. It does not output the three INSERT statements following USE test, but does output the three INSERT statements following USE db2.

Row-based logging. mysqlbinlog outputs only entries that change tables belonging to db_name. The default database has no effect on this. Suppose that the binary log just described was created using row-based logging rather than statement-based logging. mysqlbinlog --database=test outputs only those entries that modify t1 in the test database, regardless of whether USE was issued or what the default database is. If a server is running with binlog_format set to MIXED and you want it to be possible to use mysqlbinlog with the --database option, you must ensure that tables that are modified are in the database selected by USE. (In particular, no cross-database updates should be used.)

When used together with the --rewrite-db option, the --rewrite-db option is applied first; then the --database option is applied, using the rewritten database name. The order in which the options are provided makes no difference in this regard.

? --debug[=debug_options], -# [debug_options] Write a debugging log.

A typical debug_options string is d:t:o,file_name. The default is d:t:o,/tmp/mysqlbinlog.trace.

This option is available only if MySQL was built using WITH_DEBUG. MySQL release binaries provided by Oracle are not built using this option.

? `--debug-check` Print some debugging information when the program exits.

This option is available only if MySQL was built using `WITH_DEBUG`. MySQL release binaries provided by Oracle are not built using this option.

? `--debug-info` Print debugging information and memory and CPU usage statistics when the program exits.

This option is available only if MySQL was built using `WITH_DEBUG`. MySQL release binaries provided by Oracle are not built using this option.

? `--default-auth=plugin` A hint about which client-side authentication plugin to use. See Section 6.2.17, [?Pluggable Authentication?](#).

? `--defaults-extra-file=file_name` Read this option file after the global option file but (on Unix) before the user option file. If the file does not exist or is otherwise inaccessible, an error occurs. If `file_name` is not an absolute path name, it is interpreted relative to the current directory.

For additional information about this and other option-file options, see Section 4.2.2.3, [?Command-Line Options that Affect Option-File Handling?](#).

? `--defaults-file=file_name` Use only the given option file. If the file does not exist or is otherwise inaccessible, an error occurs. If `file_name` is not an absolute path name, it is interpreted relative to the current directory.

Exception: Even with `--defaults-file`, client programs read `.mylogin.cnf`.

For additional information about this and other option-file options, see Section 4.2.2.3, [?Command-Line Options that Affect Option-File Handling?](#).

? `--defaults-group-suffix=str` Read not only the usual option groups, but also groups with the usual names and a suffix of `str`. For example, `mysqlbinlog` normally reads the `[client]` and `[mysqlbinlog]` groups. If this option is given as `--defaults-group-suffix=_other`,

mysqlbinlog also reads the [client_other] and [mysqlbinlog_other] groups.

For additional information about this and other option-file options, see Section 4.2.2.3, "Command-Line Options that Affect Option-File Handling".

? --disable-log-bin, -D Disable binary logging. This is useful for avoiding an endless loop if you use the --to-last-log option and are sending the output to the same MySQL server. This option also is useful when restoring after an unexpected exit to avoid duplication of the statements you have logged.

This option causes mysqlbinlog to include a SET sql_log_bin = 0 statement in its output to disable binary logging of the remaining output. Manipulating the session value of the sql_log_bin system variable is a restricted operation, so this option requires that you have privileges sufficient to set restricted session variables. See Section 5.1.9.1, "System Variable Privileges".

? --exclude-gtids=gtid_set Do not display any of the groups listed in the gtid_set.

? --force-if-open, -F Read binary log files even if they are open or were not closed properly (IN_USE flag is set); do not fail if the file ends with a truncated event.

The IN_USE flag is set only for the binary log that is currently written by the server; if the server has crashed, the flag remains set until the server is started up again and recovers the binary log. Without this option, mysqlbinlog refuses to process a file with this flag set. Since the server may be in the process of writing the file, truncation of the last event is considered normal.

? --force-read, -f With this option, if mysqlbinlog reads a binary log event that it does not recognize, it prints a warning, ignores the event, and continues. Without this option, mysqlbinlog stops if it reads such an event.

? --get-server-public-key Request from the server the public key

required for RSA key pair-based password exchange. This option applies to clients that authenticate with the `caching_sha2_password` authentication plugin. For that plugin, the server does not send the public key unless requested. This option is ignored for accounts that do not authenticate with that plugin. It is also ignored if RSA-based password exchange is not used, as is the case when the client connects to the server using a secure connection.

If `--server-public-key-path=file_name` is given and specifies a valid public key file, it takes precedence over `--get-server-public-key`.

For information about the `caching_sha2_password` plugin, see Section 6.4.1.2, [?Caching SHA-2 Pluggable Authentication?](#).

? `--hexdump, -H` Display a hex dump of the log in comments, as described in the section called [?MYSQLBINLOG HEX DUMP FORMAT?](#). The hex output can be helpful for replication debugging.

? `--host=host_name, -h host_name` Get the binary log from the MySQL server on the given host.

? `--idempotent` Tell the MySQL Server to use idempotent mode while processing updates; this causes suppression of any duplicate-key or key-not-found errors that the server encounters in the current session while processing updates. This option may prove useful whenever it is desirable or necessary to replay one or more binary logs to a MySQL Server which may not contain all of the data to which the logs refer.

The scope of effect for this option includes the current `mysqlbinlog` client and session only.

? `--include-gtids=gtid_set` Display only the groups listed in the `gtid_set`.

? `--local-load=dir_name, -l dir_name` For data loading operations corresponding to `LOAD DATA` statements, `mysqlbinlog` extracts the files from the binary log events, writes them as temporary files to the local file system, and writes `LOAD DATA LOCAL` statements to cause the files to be loaded. By default, `mysqlbinlog` writes these

temporary files to an operating system-specific directory. The `--local-load` option can be used to explicitly specify the directory where `mysqlbinlog` should prepare local temporary files. Because other processes can write files to the default system-specific directory, it is advisable to specify the `--local-load` option to `mysqlbinlog` to designate a different directory for data files, and then designate that same directory by specifying the `--load-data-local-dir` option to `mysql` when processing the output from `mysqlbinlog`. For example:

```
mysqlbinlog --local-load=/my/local/data ...  
| mysql --load-data-local-dir=/my/local/data ...
```

Important

These temporary files are not automatically removed by `mysqlbinlog` or any other MySQL program.

? `--login-path=name` Read options from the named login path in the `.mylogin.cnf` login path file. A `?login path?` is an option group containing options that specify which MySQL server to connect to and which account to authenticate as. To create or modify a login path file, use the `mysql_config_editor` utility. See `mysql_config_editor(1)`.

For additional information about this and other option-file options, see Section 4.2.2.3, `?Command-Line Options that Affect Option-File Handling?`.

? `--no-defaults` Do not read any option files. If program startup fails due to reading unknown options from an option file, `--no-defaults` can be used to prevent them from being read. The exception is that the `.mylogin.cnf` file is read in all cases, if it exists. This permits passwords to be specified in a safer way than on the command line even when `--no-defaults` is used. To create `.mylogin.cnf`, use the `mysql_config_editor` utility. See `mysql_config_editor(1)`.

For additional information about this and other option-file options, see Section 4.2.2.3, `?Command-Line Options that Affect`

Option-File Handling?

- ? --offset=N, -o N Skip the first N entries in the log.
- ? --open-files-limit=N Specify the number of open file descriptors to reserve.
- ? --password[=password], -p[password] The password of the MySQL account used for connecting to the server. The password value is optional. If not given, mysqlbinlog prompts for one. If given, there must be no space between --password= or -p and the password following it. If no password option is specified, the default is to send no password.

Specifying a password on the command line should be considered insecure. To avoid giving the password on the command line, use an option file. See Section 6.1.2.1, "End-User Guidelines for Password Security".

To explicitly specify that there is no password and that mysqlbinlog should not prompt for one, use the --skip-password option.

- ? --plugin-dir=dir_name The directory in which to look for plugins. Specify this option if the --default-auth option is used to specify an authentication plugin but mysqlbinlog does not find it. See Section 6.2.17, "Pluggable Authentication".
- ? --port=port_num, -P port_num The TCP/IP port number to use for connecting to a remote server.
- ? --print-defaults Print the program name and all options that it gets from option files.

For additional information about this and other option-file options, see Section 4.2.2.3, "Command-Line Options that Affect Option-File Handling".
- ? --print-table-metadata Print table related metadata from the binary log. Configure the amount of table related metadata binary logged using binlog-row-metadata.
- ? --protocol={TCP|SOCKET|PIPE|MEMORY} The transport protocol to use for connecting to the server. It is useful when the other

connection parameters normally result in use of a protocol other than the one you want. For details on the permissible values, see Section 4.2.7, "Connection Transport Protocols".

? --raw By default, mysqlbinlog reads binary log files and writes events in text format. The --raw option tells mysqlbinlog to write them in their original binary format. Its use requires that --read-from-remote-server also be used because the files are requested from a server. mysqlbinlog writes one output file for each file read from the server. The --raw option can be used to make a backup of a server's binary log. With the --stop-never option, the backup is "live" because mysqlbinlog stays connected to the server. By default, output files are written in the current directory with the same names as the original log files. Output file names can be modified using the --result-file option. For more information, see the section called "USING MYSQLBINLOG TO BACK UP BINARY LOG FILES".

? --read-from-remote-source=type From MySQL 8.0.26, use --read-from-remote-source, and before MySQL 8.0.26, use --read-from-remote-master. Both options have the same effect. The options read binary logs from a MySQL server with the COM_BINLOG_DUMP or COM_BINLOG_DUMP_GTID commands by setting the option value to either BINLOG-DUMP-NON-GTIDS or BINLOG-DUMP-GTIDS, respectively. If --read-from-remote-source=BINLOG-DUMP-GTIDS or --read-from-remote-master=BINLOG-DUMP-GTIDS is combined with --exclude-gtids, transactions can be filtered out on the source, avoiding unnecessary network traffic.

The connection parameter options are used with these options or the --read-from-remote-server option. These options are --host, --password, --port, --protocol, --socket, and --user. If none of the remote options is specified, the connection parameter options are ignored.

The REPLICATION SLAVE privilege is required to use these options.

? --read-from-remote-master=type Use this option before MySQL 8.0.26

rather than `--read-from-remote-source`. Both options have the same effect.

? `--read-from-remote-server=file_name, -R` Read the binary log from a MySQL server rather than reading a local log file. This option requires that the remote server be running. It works only for binary log files on the remote server, not relay log files, and takes only the binary log file name (including the numeric suffix) as its argument, while ignoring any path.

The connection parameter options are used with this option or the `--read-from-remote-master` option. These options are `--host`, `--password`, `--port`, `--protocol`, `--socket`, and `--user`. If neither of the remote options is specified, the connection parameter options are ignored.

The `REPLICATION SLAVE` privilege is required to use this option.

This option is like

`--read-from-remote-master=BINLOG-DUMP-NON-GTIDS`.

? `--result-file=name, -r name` Without the `--raw` option, this option indicates the file to which `mysqlbinlog` writes text output. With `--raw`, `mysqlbinlog` writes one binary output file for each log file transferred from the server, writing them by default in the current directory using the same names as the original log file. In this case, the `--result-file` option value is treated as a prefix that modifies output file names.

? `--require-row-format` Require row-based binary logging format for events. This option enforces row-based replication events for `mysqlbinlog` output. The stream of events produced with this option would be accepted by a replication channel that is secured using the `REQUIRE_ROW_FORMAT` option of the `CHANGE REPLICATION SOURCE TO` statement (from MySQL 8.0.23) or `CHANGE MASTER TO` statement (before MySQL 8.0.23). `binlog_format=ROW` must be set on the server where the binary log was written. When you specify this option, `mysqlbinlog` stops with an error message if it encounters any events that are disallowed under the `REQUIRE_ROW_FORMAT` restrictions,

including LOAD DATA INFILE instructions, creating or dropping temporary tables, INTVAR, RAND, or USER_VAR events, and non-row-based events within a DML transaction. mysqlbinlog also prints a SET @@session.require_row_format statement at the start of its output to apply the restrictions when the output is executed, and does not print the SET @@session.pseudo_thread_id statement. This option was added in MySQL 8.0.19.

? --rewrite-db='from_name->to_name' When reading from a row-based or statement-based log, rewrite all occurrences of from_name to to_name. Rewriting is done on the rows, for row-based logs, as well as on the USE clauses, for statement-based logs.

Warning

Statements in which table names are qualified with database names are not rewritten to use the new name when using this option.

The rewrite rule employed as a value for this option is a string having the form 'from_name->to_name', as shown previously, and for this reason must be enclosed by quotation marks.

To employ multiple rewrite rules, specify the option multiple times, as shown here:

```
mysqlbinlog --rewrite-db='dbcurrent->dbold' --rewrite-db='dbtest->dbcurrent' \  
binlog.00001 > /tmp/statements.sql
```

When used together with the --database option, the --rewrite-db option is applied first; then --database option is applied, using the rewritten database name. The order in which the options are provided makes no difference in this regard.

This means that, for example, if mysqlbinlog is started with --rewrite-db='mydb->yourdb' --database=yourdb, then all updates to any tables in databases mydb and yourdb are included in the output.

On the other hand, if it is started with --rewrite-db='mydb->yourdb' --database=mydb, then mysqlbinlog outputs no statements at all: since all updates to mydb are first rewritten as updates to yourdb before applying the --database

option, there remain no updates that match `--database=mydb`.

? `--server-id=id` Display only those events created by the server having the given server ID.

? `--server-id-bits=N` Use only the first N bits of the `server_id` to identify the server. If the binary log was written by a `mysqld` with `server-id-bits` set to less than 32 and user data stored in the most significant bit, running `mysqlbinlog` with `--server-id-bits` set to 32 enables this data to be seen.

This option is supported only by the version of `mysqlbinlog` supplied with the NDB Cluster distribution, or built with NDB Cluster support.

? `--server-public-key-path=file_name` The path name to a file in PEM format containing a client-side copy of the public key required by the server for RSA key pair-based password exchange. This option applies to clients that authenticate with the `sha256_password` or `caching_sha2_password` authentication plugin. This option is ignored for accounts that do not authenticate with one of those plugins. It is also ignored if RSA-based password exchange is not used, as is the case when the client connects to the server using a secure connection.

If `--server-public-key-path=file_name` is given and specifies a valid public key file, it takes precedence over `--get-server-public-key`.

For `sha256_password`, this option applies only if MySQL was built using OpenSSL.

For information about the `sha256_password` and `caching_sha2_password` plugins, see Section 6.4.1.3, [?SHA-256 Pluggable Authentication?](#), and Section 6.4.1.2, [?Caching SHA-2 Pluggable Authentication?](#).

? `--set-charset=charset_name` Add a `SET NAMES charset_name` statement to the output to specify the character set to be used for processing log files.

? `--shared-memory-base-name=name` On Windows, the shared-memory name to use for connections made using shared memory to a local server.

The default value is MYSQL. The shared-memory name is case-sensitive.

This option applies only if the server was started with the `shared_memory` system variable enabled to support shared-memory connections.

? `--short-form, -s` Display only the statements contained in the log, without any extra information or row-based events. This is for testing only, and should not be used in production systems. It is deprecated, and you should expect it to be removed in a future release.

? `--skip-gtids[=(true|false)]` Do not include the GTIDs from the binary log files in the output dump file. For example:

```
mysqlbinlog --skip-gtids binlog.000001 > /tmp/dump.sql  
mysql -u root -p -e "source /tmp/dump.sql"
```

You should not normally use this option in production or in recovery, except in the specific, and rare, scenarios where the GTIDs are actively unwanted. For example, an administrator might want to duplicate selected transactions (such as table definitions) from a deployment to another, unrelated, deployment that will not replicate to or from the original. In that scenario, `--skip-gtids` can be used to enable the administrator to apply the transactions as if they were new, and ensure that the deployments remain unrelated. However, you should only use this option if the inclusion of the GTIDs causes a known issue for your use case.

? `--socket=path, -S path` For connections to localhost, the Unix socket file to use, or, on Windows, the name of the named pipe to use.

On Windows, this option applies only if the server was started with the `named_pipe` system variable enabled to support named-pipe connections. In addition, the user making the connection must be a member of the Windows group specified by the `named_pipe_full_access_group` system variable.

? `--ssl*` Options that begin with `--ssl` specify whether to connect to

the server using encryption and indicate where to find SSL keys and certificates. See the section called "Command Options for Encrypted Connections".

? `--ssl-fips-mode={OFF|ON|STRICT}` Controls whether to enable FIPS mode on the client side. The `--ssl-fips-mode` option differs from other `--ssl-xxx` options in that it is not used to establish encrypted connections, but rather to affect which cryptographic operations to permit. See Section 6.8, "FIPS Support".

These `--ssl-fips-mode` values are permitted:

? OFF: Disable FIPS mode.

? ON: Enable FIPS mode.

? STRICT: Enable "strict" FIPS mode.

Note

If the OpenSSL FIPS Object Module is not available, the only permitted value for `--ssl-fips-mode` is OFF. In this case, setting `--ssl-fips-mode` to ON or STRICT causes the client to produce a warning at startup and to operate in non-FIPS mode.

As of MySQL 8.0.34, this option is deprecated. Expect it to be removed in a future version of MySQL.

? `--start-datetime=datetime` Start reading the binary log at the first event having a timestamp equal to or later than the `datetime` argument. The `datetime` value is relative to the local time zone on the machine where you run `mysqlbinlog`. The value should be in a format accepted for the DATETIME or TIMESTAMP data types. For example:

```
mysqlbinlog --start-datetime="2005-12-25 11:25:56" binlog.000003
```

This option is useful for point-in-time recovery. See Section 7.5, "Point-in-Time (Incremental) Recovery".

? `--start-position=N, -j N` Start decoding the binary log at the log position N, including in the output any events that begin at position N or after. The position is a byte point in the log file, not an event counter; it needs to point to the starting position of an event to generate useful output. This option applies to the

first log file named on the command line.

Prior to MySQL 8.0.33, the maximum value supported for this option was 4294967295 (232-1). In MySQL 8.0.33 and later, it is 18446744073709551616 (264-1), unless `--read-from-remote-server` or `--read-from-remote-source` is also used, in which case the maximum is 4294967295.

This option is useful for point-in-time recovery. See Section 7.5, [Point-in-Time \(Incremental\) Recovery](#).

? `--stop-datetime=datetime` Stop reading the binary log at the first event having a timestamp equal to or later than the `datetime` argument. See the description of the `--start-datetime` option for information about the `datetime` value.

This option is useful for point-in-time recovery. See Section 7.5, [Point-in-Time \(Incremental\) Recovery](#).

? `--stop-never` This option is used with `--read-from-remote-server`. It tells `mysqlbinlog` to remain connected to the server. Otherwise `mysqlbinlog` exits when the last log file has been transferred from the server. `--stop-never` implies `--to-last-log`, so only the first log file to transfer need be named on the command line.

`--stop-never` is commonly used with `--raw` to make a live binary log backup, but also can be used without `--raw` to maintain a continuous text display of log events as the server generates them.

With `--stop-never`, by default, `mysqlbinlog` reports a server ID of 1 when it connects to the server. Use `--connection-server-id` to explicitly specify an alternative ID to report. It can be used to avoid a conflict with the ID of a replica server or another `mysqlbinlog` process. See the section called [SPECIFYING THE MYSQLBINLOG SERVER ID](#).

? `--stop-never-slave-server-id=id` This option is deprecated; expect it to be removed in a future release. Use the `--connection-server-id` option instead to specify a server ID for `mysqlbinlog` to report.

? `--stop-position=N` Stop decoding the binary log at the log position

N, excluding from the output any events that begin at position N or after. The position is a byte point in the log file, not an event counter; it needs to point to a spot after the starting position of the last event you want to include in the output. The event starting before position N and finishing at or after the position is the last event to be processed. This option applies to the last log file named on the command line.

This option is useful for point-in-time recovery. See Section 7.5, "Point-in-Time (Incremental) Recovery".

? `--tls-ciphersuites=ciphersuite_list` The permissible ciphersuites for encrypted connections that use TLSv1.3. The value is a list of one or more colon-separated ciphersuite names. The ciphersuites that can be named for this option depend on the SSL library used to compile MySQL. For details, see Section 6.3.2, "Encrypted Connection TLS Protocols and Ciphers".

This option was added in MySQL 8.0.16.

? `--tls-version=protocol_list` The permissible TLS protocols for encrypted connections. The value is a list of one or more comma-separated protocol names. The protocols that can be named for this option depend on the SSL library used to compile MySQL. For details, see Section 6.3.2, "Encrypted Connection TLS Protocols and Ciphers".

? `--to-last-log, -t` Do not stop at the end of the requested binary log from a MySQL server, but rather continue printing until the end of the last binary log. If you send the output to the same MySQL server, this may lead to an endless loop. This option requires `--read-from-remote-server`.

? `--user=user_name, -u user_name` The user name of the MySQL account to use when connecting to a remote server.

If you are using the Rewriter plugin with MySQL 8.0.31 or later, you should grant this user the `SKIP_QUERY_REWRITE` privilege.

? `--verbose, -v` Reconstruct row events and display them as commented SQL statements, with table partition information where applicable.

If this option is given twice (by passing in either "-vv" or "--verbose --verbose"), the output includes comments to indicate column data types and some metadata, and informational log events such as row query log events if the `binlog_rows_query_log_events` system variable is set to TRUE.

For examples that show the effect of `--base64-output` and `--verbose` on row event output, see the section called `?MYSQLBINLOG ROW EVENT DISPLAY?`.

? `--verify-binlog-checksum, -c` Verify checksums in binary log files.

? `--version, -V` Display version information and exit.

The `mysqlbinlog` version number shown when using this option is 3.4.

? `--zstd-compression-level=level` The compression level to use for connections to the server that use the zstd compression algorithm.

The permitted levels are from 1 to 22, with larger values indicating increasing levels of compression. The default zstd compression level is 3. The compression level setting has no effect on connections that do not use zstd compression.

For more information, see Section 4.2.8, `?Connection Compression Control?`.

This option was added in MySQL 8.0.18.

You can pipe the output of `mysqlbinlog` into the `mysql` client to execute the events contained in the binary log. This technique is used to recover from an unexpected exit when you have an old backup (see Section 7.5, `?Point-in-Time (Incremental) Recovery?`). For example:

```
mysqlbinlog binlog.000001 | mysql -u root -p
```

Or:

```
mysqlbinlog binlog.[0-9]* | mysql -u root -p
```

If the statements produced by `mysqlbinlog` may contain BLOB values, these may cause problems when `mysql` processes them. In this case, invoke `mysql` with the `--binary-mode` option.

You can also redirect the output of `mysqlbinlog` to a text file instead, if you need to modify the statement log first (for example, to remove statements that you do not want to execute for some reason). After

editing the file, execute the statements that it contains by using it as input to the mysql program:

```
mysqlbinlog binlog.000001 > tmpfile
```

```
... edit tmpfile ...
```

```
mysql -u root -p < tmpfile
```

When mysqlbinlog is invoked with the --start-position option, it displays only those events with an offset in the binary log greater than or equal to a given position (the given position must match the start of one event). It also has options to stop and start when it sees an event with a given date and time. This enables you to perform point-in-time recovery using the --stop-datetime option (to be able to say, for example, ?roll forward my databases to how they were today at 10:30 a.m.?).

Processing multiple files. If you have more than one binary log to execute on the MySQL server, the safe method is to process them all using a single connection to the server. Here is an example that demonstrates what may be unsafe:

```
mysqlbinlog binlog.000001 | mysql -u root -p # DANGER!!
```

```
mysqlbinlog binlog.000002 | mysql -u root -p # DANGER!!
```

Processing binary logs this way using multiple connections to the server causes problems if the first log file contains a CREATE TEMPORARY TABLE statement and the second log contains a statement that uses the temporary table. When the first mysql process terminates, the server drops the temporary table. When the second mysql process attempts to use the table, the server reports ?unknown table.?

To avoid problems like this, use a single mysql process to execute the contents of all binary logs that you want to process. Here is one way to do so:

```
mysqlbinlog binlog.000001 binlog.000002 | mysql -u root -p
```

Another approach is to write all the logs to a single file and then process the file:

```
mysqlbinlog binlog.000001 > /tmp/statements.sql
```

```
mysqlbinlog binlog.000002 >> /tmp/statements.sql
```

```
mysql -u root -p -e "source /tmp/statements.sql"
```

From MySQL 8.0.12, you can also supply multiple binary log files to `mysqlbinlog` as streamed input using a shell pipe. An archive of compressed binary log files can be decompressed and provided directly to `mysqlbinlog`. In this example, `binlog-files_1.gz` contains multiple binary log files for processing. The pipeline extracts the contents of `binlog-files_1.gz`, pipes the binary log files to `mysqlbinlog` as standard input, and pipes the output of `mysqlbinlog` into the `mysql` client for execution:

```
gzip -cd binlog-files_1.gz | ./mysqlbinlog - | ./mysql -uroot -p
```

You can specify more than one archive file, for example:

```
gzip -cd binlog-files_1.gz binlog-files_2.gz | ./mysqlbinlog - | ./mysql -uroot -p
```

For streamed input, do not use `--stop-position`, because `mysqlbinlog` cannot identify the last log file to apply this option.

LOAD DATA operations. `mysqlbinlog` can produce output that reproduces a `LOAD DATA` operation without the original data file. `mysqlbinlog` copies the data to a temporary file and writes a `LOAD DATA LOCAL` statement that refers to the file. The default location of the directory where these files are written is system-specific. To specify a directory explicitly, use the `--local-load` option.

Because `mysqlbinlog` converts `LOAD DATA` statements to `LOAD DATA LOCAL` statements (that is, it adds `LOCAL`), both the client and the server that you use to process the statements must be configured with the `LOCAL` capability enabled. See Section 6.1.6, "Security Considerations for `LOAD DATA LOCAL`".

Warning

The temporary files created for `LOAD DATA LOCAL` statements are not automatically deleted because they are needed until you actually execute those statements. You should delete the temporary files yourself after you no longer need the statement log. The files can be found in the temporary file directory and have names like `original_file_name-#-#`.

The --hexdump option causes mysqlbinlog to produce a hex dump of the binary log contents:

```
mysqlbinlog --hexdump source-bin.000001
```

The hex output consists of comment lines beginning with #, so the output might look like this for the preceding command:

```
/*!40019 SET @@SESSION.max_insert_delayed_threads=0*/;
/*!50003 SET @OLD_COMPLETION_TYPE=@@COMPLETION_TYPE,COMPLETION_TYPE=0*/;
# at 4
#051024 17:24:13 server id 1  end_log_pos 98
# Position Timestamp Type Master ID Size Master Pos Flags
# 00000004 9d fc 5c 43 0f 01 00 00 00 5e 00 00 00 62 00 00 00 00 00
# 00000017 04 00 35 2e 30 2e 31 35 2d 64 65 62 75 67 2d 6c |..5.0.15.debug.|
# 00000027 6f 67 00 00 00 00 00 00 00 00 00 00 00 00 00 |og.....|
# 00000037 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 |.....|
# 00000047 00 00 00 00 9d fc 5c 43 13 38 0d 00 08 00 12 00 |.....C.8.....|
# 00000057 04 04 04 04 12 00 00 4b 00 04 1a |.....K...|
# Start: binlog v 4, server v 5.0.15-debug-log created 051024 17:24:13
# at startup
ROLLBACK;
```

Hex dump output currently contains the elements in the following list.

This format is subject to change. For more information about binary log format, see MySQL Internals: The Binary Log[1].

- ? Position: The byte position within the log file.
- ? Timestamp: The event timestamp. In the example shown, '9d fc 5c 43' is the representation of '051024 17:24:13' in hexadecimal.
- ? Type: The event type code.
- ? Master ID: The server ID of the replication source server that created the event.
- ? Size: The size in bytes of the event.
- ? Master Pos: The position of the next event in the original source's binary log file.
- ? Flags: Event flag values.

The following examples illustrate how mysqlbinlog displays row events that specify data modifications. These correspond to events with the WRITE_ROWS_EVENT, UPDATE_ROWS_EVENT, and DELETE_ROWS_EVENT type codes. The --base64-output=DECODE-ROWS and --verbose options may be used to affect row event output.

Suppose that the server is using row-based binary logging and that you execute the following sequence of statements:

```
CREATE TABLE t
(
  id INT NOT NULL,
  name VARCHAR(20) NOT NULL,
  date DATE NULL
) ENGINE = InnoDB;
START TRANSACTION;
INSERT INTO t VALUES(1, 'apple', NULL);
UPDATE t SET name = 'pear', date = '2009-01-01' WHERE id = 1;
DELETE FROM t WHERE id = 1;
COMMIT;
```

By default, mysqlbinlog displays row events encoded as base-64 strings using BINLOG statements. Omitting extraneous lines, the output for the row events produced by the preceding statement sequence looks like this:

```
$> mysqlbinlog log_file
...
# at 218
#080828 15:03:08 server id 1  end_log_pos 258  Write_rows: table id 17 flags: STMT_END_F
BINLOG '
fAS3SBMBAAAALAAAANoAAAAAABEAAAAAAAAAABHRic3QAAXQAAwMPCgIUAAQ=
fAS3SBcBAAAAKAAAAAIBAAAQABEAAAAAAAAEAA//8AQAAAVhcHBsZQ==
/*!*/;
...
# at 302
#080828 15:03:08 server id 1  end_log_pos 356  Update_rows: table id 17 flags: STMT_END_F
```

```
BINLOG '
fAS3SBMBAAAALAAAAC4BAAAAABEAAAAAAAAABHRic3QAAXQAAwMPCgIUAAQ=
fAS3SBgBAAAANgAAAGQBAAAQABEAAAAAAAAEAA////AEAAAFYXBwbGX4AQAAARwZWFyIbIP
/*!*/;
```

...

at 400

#080828 15:03:08 server id 1 end_log_pos 442 Delete_rows: table id 17 flags: STMT_END_F

```
BINLOG '
fAS3SBMBAAAALAAAABAAAAABEAAAAAAAAABHRic3QAAXQAAwMPCgIUAAQ=
fAS3SBkBAAAAKgAAALoBAAAQABEAAAAAAAAEAA//4AQAAARwZWFyIbIP
/*!*/;
```

To see the row events as comments in the form of ?pseudo-SQL? statements, run mysqlbinlog with the --verbose or -v option. This output level also shows table partition information where applicable.

The output contains lines beginning with ###:

```
$> mysqlbinlog -v log_file
```

...

at 218

#080828 15:03:08 server id 1 end_log_pos 258 Write_rows: table id 17 flags: STMT_END_F

```
BINLOG '
fAS3SBMBAAAALAAAANoAAAAABEAAAAAAAAABHRic3QAAXQAAwMPCgIUAAQ=
fAS3SBcBAAAAKAAAAIBAAAQABEAAAAAAAAEAA//8AQAAAVhcHBsZQ==
/*!*/;
```

```
### INSERT INTO test.t
```

```
### SET
```

```
### @1=1
```

```
### @2='apple'
```

```
### @3=NULL
```

...

at 302

#080828 15:03:08 server id 1 end_log_pos 356 Update_rows: table id 17 flags: STMT_END_F

```
BINLOG '
fAS3SBMBAAAALAAAAC4BAAAAABEAAAAAAAAABHRic3QAAXQAAwMPCgIUAAQ=
```

```
fAS3SBgBAAAANgAAAGQBAAAQABEAAAAAAAAEAA////AEAAAFYXBwbGX4AQAAARwZWFyIbIP
```

```
/*!*/;
```

```
### UPDATE test.t
```

```
### WHERE
```

```
### @1=1
```

```
### @2='apple'
```

```
### @3=NULL
```

```
### SET
```

```
### @1=1
```

```
### @2='pear'
```

```
### @3='2009:01:01'
```

```
...
```

```
# at 400
```

```
#080828 15:03:08 server id 1 end_log_pos 442 Delete_rows: table id 17 flags: STMT_END_F
```

```
BINLOG '
```

```
fAS3SBMBAAAALAAAAJABAAAABEAAAAAAAAABHRlc3QAAXQAAwMPCglUAAQ=
```

```
fAS3SBkBAAAAGgAAALoBAAAQABEAAAAAAAAEAA//4AQAAARwZWFyIbIP
```

```
/*!*/;
```

```
### DELETE FROM test.t
```

```
### WHERE
```

```
### @1=1
```

```
### @2='pear'
```

```
### @3='2009:01:01'
```

Specify --verbose or -v twice to also display data types and some metadata for each column, and informational log events such as row query log events if the binlog_rows_query_log_events system variable is set to TRUE. The output contains an additional comment following each column change:

```
$> mysqlbinlog -vv log_file
```

```
...
```

```
# at 218
```

```
#080828 15:03:08 server id 1 end_log_pos 258 Write_rows: table id 17 flags: STMT_END_F
```

```
BINLOG '
```

```
fAS3SBMBAAAALAAAANoAAAAAABEAAAAAAAAABHRic3QAAXQAAwMPCgIUAAQ=
fAS3SBcBAAAAKAAAAAIBAAAQABEAAAAAAAAEAA//8AQAAAAVhcHBsZQ==
/*!*/;
### INSERT INTO test.t
### SET
### @1=1 /* INT meta=0 nullable=0 is_null=0 */
### @2='apple' /* VARSTRING(20) meta=20 nullable=0 is_null=0 */
### @3=NULL /* VARSTRING(20) meta=0 nullable=1 is_null=1 */
...
# at 302
#080828 15:03:08 server id 1 end_log_pos 356 Update_rows: table id 17 flags: STMT_END_F
BINLOG '
fAS3SBMBAAAALAAAAC4BAAAAABEAAAAAAAAABHRic3QAAXQAAwMPCgIUAAQ=
fAS3SBgBAAAANgAAAGQBAAAQABEAAAAAAAAEAA////AEAAAFYXBwbGX4AQAAARwZWFyIbIP
/*!*/;
### UPDATE test.t
### WHERE
### @1=1 /* INT meta=0 nullable=0 is_null=0 */
### @2='apple' /* VARSTRING(20) meta=20 nullable=0 is_null=0 */
### @3=NULL /* VARSTRING(20) meta=0 nullable=1 is_null=1 */
### SET
### @1=1 /* INT meta=0 nullable=0 is_null=0 */
### @2='pear' /* VARSTRING(20) meta=20 nullable=0 is_null=0 */
### @3='2009:01:01' /* DATE meta=0 nullable=1 is_null=0 */
...
# at 400
#080828 15:03:08 server id 1 end_log_pos 442 Delete_rows: table id 17 flags: STMT_END_F
BINLOG '
fAS3SBMBAAAALAAAAJABAAAAABEAAAAAAAAABHRic3QAAXQAAwMPCgIUAAQ=
fAS3SBkBAAAAKgAAALoBAAAQABEAAAAAAAAEAA//4AQAAARwZWFyIbIP
/*!*/;
### DELETE FROM test.t
### WHERE
```

```
#### @1=1 /* INT meta=0 nullable=0 is_null=0 */
```

```
#### @2='pear' /* VARSTRING(20) meta=20 nullable=0 is_null=0 */
```

```
#### @3='2009:01:01' /* DATE meta=0 nullable=1 is_null=0 */
```

You can tell mysqlbinlog to suppress the BINLOG statements for row events by using the `--base64-output=DECODE-ROWS` option. This is similar to `--base64-output=NEVER` but does not exit with an error if a row event is found. The combination of `--base64-output=DECODE-ROWS` and `--verbose` provides a convenient way to see row events only as SQL statements:

```
$> mysqlbinlog -v --base64-output=DECODE-ROWS log_file
```

```
...
```

```
# at 218
```

```
#080828 15:03:08 server id 1 end_log_pos 258 Write_rows: table id 17 flags: STMT_END_F
```

```
#### INSERT INTO test.t
```

```
#### SET
```

```
#### @1=1
```

```
#### @2='apple'
```

```
#### @3=NULL
```

```
...
```

```
# at 302
```

```
#080828 15:03:08 server id 1 end_log_pos 356 Update_rows: table id 17 flags: STMT_END_F
```

```
#### UPDATE test.t
```

```
#### WHERE
```

```
#### @1=1
```

```
#### @2='apple'
```

```
#### @3=NULL
```

```
#### SET
```

```
#### @1=1
```

```
#### @2='pear'
```

```
#### @3='2009:01:01'
```

```
...
```

```
# at 400
```

```
#080828 15:03:08 server id 1 end_log_pos 442 Delete_rows: table id 17 flags: STMT_END_F
```

```
#### DELETE FROM test.t
```

```
### WHERE
```

```
### @1=1
```

```
### @2='pear'
```

```
### @3='2009:01:01'
```

Note

You should not suppress BINLOG statements if you intend to re-execute mysqlbinlog output.

The SQL statements produced by --verbose for row events are much more readable than the corresponding BINLOG statements. However, they do not correspond exactly to the original SQL statements that generated the events. The following limitations apply:

- ? The original column names are lost and replaced by @N, where N is a column number.
- ? Character set information is not available in the binary log, which affects string column display:
 - ? There is no distinction made between corresponding binary and nonbinary string types (BINARY and CHAR, VARBINARY and VARCHAR, BLOB and TEXT). The output uses a data type of STRING for fixed-length strings and VARSTRING for variable-length strings.
 - ? For multibyte character sets, the maximum number of bytes per character is not present in the binary log, so the length for string types is displayed in bytes rather than in characters. For example, STRING(4) is used as the data type for values from either of these column types:
 - CHAR(4) CHARACTER SET latin1
 - CHAR(2) CHARACTER SET ucs2
- ? Due to the storage format for events of type UPDATE_ROWS_EVENT, UPDATE statements are displayed with the WHERE clause preceding the SET clause.

Proper interpretation of row events requires the information from the format description event at the beginning of the binary log. Because mysqlbinlog does not know in advance whether the rest of the log contains row events, by default it displays the format description

event using a BINLOG statement in the initial part of the output.

If the binary log is known not to contain any events requiring a BINLOG statement (that is, no row events), the `--base64-output=NEVER` option can be used to prevent this header from being written.

USING MYSQLBINLOG TO BACK UP BINARY LOG FILES

By default, `mysqlbinlog` reads binary log files and displays their contents in text format. This enables you to examine events within the files more easily and to re-execute them (for example, by using the output as input to `mysql`). `mysqlbinlog` can read log files directly from the local file system, or, with the `--read-from-remote-server` option, it can connect to a server and request binary log contents from that server. `mysqlbinlog` writes text output to its standard output, or to the file named as the value of the `--result-file=file_name` option if that option is given.

? `mysqlbinlog` Backup Capabilities

? `mysqlbinlog` Backup Options

? Static and Live Backups

? Output File Naming

? Example: `mysqldump` + `mysqlbinlog` for Backup and Restore

? `mysqlbinlog` Backup Restrictions

`mysqlbinlog` Backup Capabilities

`mysqlbinlog` can read binary log files and write new files containing the same content—that is, in binary format rather than text format.

This capability enables you to easily back up a binary log in its original format. `mysqlbinlog` can make a static backup, backing up a set of log files and stopping when the end of the last file is reached.

It can also make a continuous (?live?) backup, staying connected to the server when it reaches the end of the last log file and continuing to copy new events as they are generated. In continuous-backup operation, `mysqlbinlog` runs until the connection ends (for example, when the server exits) or `mysqlbinlog` is forcibly terminated. When the connection ends, `mysqlbinlog` does not wait and retry the connection, unlike a replica server. To continue a live backup after the server has

been restarted, you must also restart mysqlbinlog.

Important

mysqlbinlog can back up both encrypted and unencrypted binary log files . However, copies of encrypted binary log files that are generated using mysqlbinlog are stored in an unencrypted format.

mysqlbinlog Backup Options

Binary log backup requires that you invoke mysqlbinlog with two options at minimum:

? The --read-from-remote-server (or -R) option tells mysqlbinlog to connect to a server and request its binary log. (This is similar to a replica server connecting to its replication source server.)

? The --raw option tells mysqlbinlog to write raw (binary) output, not text output.

Along with --read-from-remote-server, it is common to specify other options: --host indicates where the server is running, and you may also need to specify connection options such as --user and --password.

Several other options are useful in conjunction with --raw:

? --stop-never: Stay connected to the server after reaching the end of the last log file and continue to read new events.

? --connection-server-id=id: The server ID that mysqlbinlog reports when it connects to a server. When --stop-never is used, the default reported server ID is 1. If this causes a conflict with the ID of a replica server or another mysqlbinlog process, use --connection-server-id to specify an alternative server ID. See the section called ?SPECIFYING THE MYSQLBINLOG SERVER ID?.

? --result-file: A prefix for output file names, as described later.

Static and Live Backups

To back up a server's binary log files with mysqlbinlog, you must specify file names that actually exist on the server. If you do not know the names, connect to the server and use the SHOW BINARY LOGS statement to see the current names. Suppose that the statement produces this output:

```
mysql> SHOW BINARY LOGS;
```

Log_name	File_size	Encrypted
binlog.000130	27459	No
binlog.000131	13719	No
binlog.000132	43268	No

With that information, you can use `mysqlbinlog` to back up the binary log to the current directory as follows (enter each command on a single line):

? To make a static backup of `binlog.000130` through `binlog.000132`, use either of these commands:

```
mysqlbinlog --read-from-remote-server --host=host_name --raw
binlog.000130 binlog.000131 binlog.000132
mysqlbinlog --read-from-remote-server --host=host_name --raw
--to-last-log binlog.000130
```

The first command specifies every file name explicitly. The second names only the first file and uses `--to-last-log` to read through the last. A difference between these commands is that if the server happens to open `binlog.000133` before `mysqlbinlog` reaches the end of `binlog.000132`, the first command does not read it, but the second command does.

? To make a live backup in which `mysqlbinlog` starts with `binlog.000130` to copy existing log files, then stays connected to copy new events as the server generates them:

```
mysqlbinlog --read-from-remote-server --host=host_name --raw
--stop-never binlog.000130
```

With `--stop-never`, it is not necessary to specify `--to-last-log` to read to the last log file because that option is implied.

Output File Naming

Without `--raw`, `mysqlbinlog` produces text output and the `--result-file` option, if given, specifies the name of the single file to which all output is written. With `--raw`, `mysqlbinlog` writes one binary output

file for each log file transferred from the server. By default, mysqlbinlog writes the files in the current directory with the same names as the original log files. To modify the output file names, use the --result-file option. In conjunction with --raw, the --result-file option value is treated as a prefix that modifies the output file names.

Suppose that a server currently has binary log files named binlog.000999 and up. If you use mysqlbinlog --raw to back up the files, the --result-file option produces output file names as shown in the following table. You can write the files to a specific directory by beginning the --result-file value with the directory path. If the --result-file value consists only of a directory name, the value must end with the pathname separator character. Output files are overwritten if they exist.

??

--result-file Option ? Output File Names ?

??

--result-file=x ? xbinlog.000999 and up ?

??

--result-file=/tmp/ ? /tmp/binlog.000999 and up ?

??

--result-file=/tmp/x ? /tmp/xbinlog.000999 and up ?

??

Example: mysqldump + mysqlbinlog for Backup and Restore

The following example describes a simple scenario that shows how to use mysqldump and mysqlbinlog together to back up a server's data and binary log, and how to use the backup to restore the server if data loss occurs. The example assumes that the server is running on host host_name and its first binary log file is named binlog.000999. Enter each command on a single line.

Use mysqlbinlog to make a continuous backup of the binary log:

```
mysqlbinlog --read-from-remote-server --host=host_name --raw  
--stop-never binlog.000999
```

Use mysqldump to create a dump file as a snapshot of the server's data.

Use --all-databases, --events, and --routines to back up all data, and --master-data=2 to include the current binary log coordinates in the dump file.

```
mysqldump --host=host_name --all-databases --events --routines --master-data=2 > dump_file
```

Execute the mysqldump command periodically to create newer snapshots as desired.

If data loss occurs (for example, if the server unexpectedly exits), use the most recent dump file to restore the data:

```
mysql --host=host_name -u root -p < dump_file
```

Then use the binary log backup to re-execute events that were written after the coordinates listed in the dump file. Suppose that the coordinates in the file look like this:

```
-- CHANGE MASTER TO MASTER_LOG_FILE='binlog.001002', MASTER_LOG_POS=27284;
```

If the most recent backed-up log file is named binlog.001004, re-execute the log events like this:

```
mysqlbinlog --start-position=27284 binlog.001002 binlog.001003 binlog.001004  
| mysql --host=host_name -u root -p
```

You might find it easier to copy the backup files (dump file and binary log files) to the server host to make it easier to perform the restore operation, or if MySQL does not allow remote root access. mysqlbinlog

Backup Restrictions

Binary log backups with mysqlbinlog are subject to these restrictions:

- ? mysqlbinlog does not automatically reconnect to the MySQL server if the connection is lost (for example, if a server restart occurs or there is a network outage).
- ? The delay for a backup is similar to the delay for a replica server.

SPECIFYING THE MYSQLBINLOG SERVER ID

When invoked with the --read-from-remote-server option, mysqlbinlog connects to a MySQL server, specifies a server ID to identify itself, and requests binary log files from the server. You can use mysqlbinlog to request log files from a server in several ways:

- ? Specify an explicitly named set of files: For each file, mysqlbinlog connects and issues a Binlog dump command. The server sends the file and disconnects. There is one connection per file.
- ? Specify the beginning file and --to-last-log: mysqlbinlog connects and issues a Binlog dump command for all files. The server sends all files and disconnects.
- ? Specify the beginning file and --stop-never (which implies --to-last-log): mysqlbinlog connects and issues a Binlog dump command for all files. The server sends all files, but does not disconnect after sending the last one.

With --read-from-remote-server only, mysqlbinlog connects using a server ID of 0, which tells the server to disconnect after sending the last requested log file.

With --read-from-remote-server and --stop-never, mysqlbinlog connects using a nonzero server ID, so the server does not disconnect after sending the last log file. The server ID is 1 by default, but this can be changed with --connection-server-id.

Thus, for the first two ways of requesting files, the server disconnects because mysqlbinlog specifies a server ID of 0. It does not disconnect if --stop-never is given because mysqlbinlog specifies a nonzero server ID.

COPYRIGHT

Copyright ? 1997, 2023, Oracle and/or its affiliates.

This documentation is free software; you can redistribute it and/or modify it only under the terms of the GNU General Public License as published by the Free Software Foundation; version 2 of the License.

This documentation is distributed in the hope that it will be useful, but WITHOUT ANY WARRANTY; without even the implied warranty of MERCHANTABILITY or FITNESS FOR A PARTICULAR PURPOSE. See the GNU General Public License for more details.

You should have received a copy of the GNU General Public License along with the program; if not, write to the Free Software Foundation, Inc., 51 Franklin Street, Fifth Floor, Boston, MA 02110-1301 USA or see

<http://www.gnu.org/licenses/>.

NOTES

1. MySQL Internals: The Binary Log

<https://dev.mysql.com/doc/internals/en/binary-log.html>

SEE ALSO

For more information, please refer to the MySQL Reference Manual, which may already be installed locally and which is also available online at <http://dev.mysql.com/doc/>.

AUTHOR

Oracle Corporation (<http://dev.mysql.com/>).

MySQL 8.0

06/02/2023

MYSQLBINLOG(1)