MySQL Enterprise Backup User's Guide (Version 3.8.2)

MySQL Enterprise Backup User's Guide (Version 3.8.2)

Abstract

This is the User's Guide for the MySQL Enterprise Backup product, the successor to the InnoDB Hot Backup product. This manual describes the procedures to back up and restore MySQL databases. It covers techniques for minimizing time and storage overhead during backups, and to keep the database available during backup operations. It illustrates the features and syntax of the `mysqlbackup` command, for example, how to back up selected databases or tables, how to back up only the changes since a previous backup, and how to transfer the backup data efficiently to a different server.

Document generated on: 2014-10-13 (revision: 5163)

Table of Contents

List of Tables

List of Examples

Preface and Legal Notices

This is the User Manual for the MySQL Enterprise Backup product.

For license information, see the Legal Notices. This product may contain third-party code. For license information on third-party code, see Appendix E, *Licenses for Third-Party Components*.

Legal Notices

Part I Getting Started with MySQL Enterprise Backup

Table of Contents

Chapter 1 Introduction to MySQL Enterprise Backup

Table of Contents

The MySQL Enterprise Backup product performs backup operations for MySQL data. It can back up all kinds of MySQL tables. It has special optimizations for fast and convenient backups of InnoDB tables. Because of the speed of InnoDB backups, and the reliability and scalability features of InnoDB tables, we recommend that you use InnoDB tables for your most important data.

This book describes the best practices regarding MySQL backups and documents how to use MySQL Enterprise Backup features to implement these practices. This book teaches you:

- Why backups are important.
- The files that make up a MySQL database and the roles they play.
- How to keep the database running during a backup.
- How to minimize the time, CPU overhead, and storage overhead for a backup job. Often, minimizing one of these aspects increases another.
- How to restore your data when disaster strikes. You learn how to verify backups and practice recovery, so that you can stay calm and confident under pressure.
- Other ways to use backup data for day-to-day administration and in deploying new servers.

1.1 Types of Backups

The various kinds of backup techniques are classified on a scale ranging from hot (the most desirable) to cold (the most disruptive). Your goal is to keep the database system, and associated applications and web sites, operating and responsive while the backup is in progress.

Hot backups are performed while the database is running. This type of backup does not block normal database operations. It captures even changes that occur while the backup is happening. For these reasons, hot backups are desirable when your database “grows up”: when the data is large enough that the backup takes significant time, and when your data is important enough to your business so that you must capture every last change, without taking your application, web site, or web service offline.

MySQL Enterprise Backup does a hot backup of all InnoDB tables. MyISAM and other non-InnoDB tables are backed up last, using the warm backup technique: the database continues to run, but the system is in a read-only state during that phase of the backup.

You can also perform cold backups while the database is stopped. To avoid service disruption, you would typically perform such a backup from a replication slave, which can be stopped without taking down the entire application or web site.

Points to Remember

To back up as much data as possible during the hot backup phase, you can designate InnoDB as the default storage engine for new tables, or convert existing tables to use the InnoDB storage engine. (In MySQL 5.5 and higher, InnoDB is now the default storage engine for new tables.)

During hot and warm backups, information about the structure of the database is retrieved automatically through a database connection. For a cold backup, you must specify file locations through configuration files or command-line options.

1.2 The mysqlbackup Command

When using the MySQL Enterprise Backup product, you primarily work with the mysqlbackup command. Based on the options you specify, this command performs all the different types of backup operations, and restore operations too. mysqlbackup can do other things that you would otherwise code into your own backup scripts, such as creating a timestamped subdirectory for each backup, compressing the backup data, and packing the backup data into a single file for easy transfer to another server.

For usage information about mysqlbackup features, see Chapter 3, *Backing Up a Database Server*. For option syntax, see Chapter 4, *mysqlbackup Command Reference*.

1.3 Overview of Backup Performance and Capacity Considerations

In your backup strategy, performance and storage space are key aspects. You want the backup to complete quickly, with little CPU overhead on the database server. You also want the backup data to be compact, so you can keep multiple backups on hand to restore at a moment's notice. Transferring the backup data to a different system should be quick and convenient. All of these aspects are controlled by options of the mysqlbackup command.

Sometimes you must balance the different kinds of overhead -- CPU cycles, storage space, and network traffic. Always be aware how much time it takes to restore the data during planned maintenance or when disaster strikes. For example, here are factors to consider for some of the key MySQL Enterprise Backup features:

- Parallel backups are the default in MySQL Enterprise Backup 3.8, a major performance improvement over earlier MySQL Enterprise Backup releases. The read, process and write are the primary sub-operations of all MEB operations. For example, in a backup operation, MySQL Enterprise Backup first reads the data from the disk, then processes this data, writes the data to disk, and reads the data again for verification. MySQL Enterprise Backup ensures that these sub-operations are independent of each other and run in parallel to gain performance improvement. Read, process and write sub-operations are performed in parallel using multiple threads of the same kind: multiple read threads, multiple process threads, and multiple write threads, resulting in better performance. The performance improvement is typically greater when RAID arrays are used as both source and target devices, and for compressed backups which can use more CPU cycles in parallel.

  Parallel backup employs block-level parallelism, using blocks of 16MB. Different threads can read, process, and write different 16MB chunks within a single file. Parallel backup improves the performance of operations whether the instance contains a single huge system tablespace, or many smaller tablespaces (represented by .ibd files created in the innodb_file_per_table mode.

- Incremental backups are faster than full backups, save storage space on the database server, and save on network traffic to transfer the backup data on a different server. Incremental backup requires additional processing to make the backup ready to restore, which you can perform on a different system to minimize CPU overhead on the database server.

- Compressed backups save on storage space for InnoDB tables, and network traffic to transfer the backup data on a different server. They do impose more CPU overhead than uncompressed backups. During restore, you need the compressed and uncompressed data at the same time, so take into account this additional storage space and the time to uncompress the data.

  In addition to compressing data within InnoDB tables, compressed backups also skip unused space within InnoDB tablespace files. Uncompressed backups include this unused space.

- When space is limited, or you have a storage device such as tape that is cheap, large, but also slow, the performance and space considerations are different. Rather than aiming for the fastest possible backup, you want to avoid storing an intermediate copy of the backup data on the database server. MySQL Enterprise Backup can produce a single-file backup and stream that file directly to the other server or device. Since the backup data is never saved to the local system, you avoid the space overhead on the database server. You also avoid the performance overhead of saving a set of backup files and then bundling them into an archive for transport to another server or storage device. For details, see Section 3.3.5.1, “Streaming the Backup Data to Another Device or Server”.

  When streaming backup data to tape, you typically do not compress the backup, because the CPU overhead on the database server to do the compression is more expensive than the additional storage space on the tape device. When streaming backup data to another server, you might compress on the original server or the destination server depending on which server has more spare CPU capacity and how much network traffic the compression could save. Or, you might leave the backup data uncompressed on the destination server so that it is ready to be restored on short notice.

For disaster recovery, when speed to restore the data is critical, you might prefer to have critical backup data already prepared and uncompressed, so that the restore operation involves as few steps as possible.

It is during a disaster recovery that speed is most critical. For example, although a logical backup performed with the `mysqldump` command might take about the same time as a physical backup with the MySQL Enterprise Backup product (at least for a small database), the MySQL Enterprise Backup restore operation is typically faster. Copying the actual data files back to the data directory skips the overhead of inserting rows and updating indexes that comes from replaying the SQL statements from `mysqldump` output.

To minimize any impact on server performance on Linux and Unix systems, MySQL Enterprise Backup writes the backup data without storing it in the operating system's disk cache, by using the `posix_fadvise()` system call. This technique minimizes any slowdown following the backup operation, by preventing frequently accessed data from being flushed from the disk cache by the large one-time read operation for the backup data.

For more on techniques and tradeoffs involving backup and restore performance, see Chapter 7, *Performance Considerations for MySQL Enterprise Backup*.

1.4 Files that Are Backed Up

DBA and development work typically involves logical structures such as tables, rows, columns, the data dictionary, and so on. For backups, you must understand the physical details of how these structures are represented by files.

Table 1.1 Files in a MySQL Enterprise Backup Output Directory

File Name, Pattern, or Extension	Relation to Original Data Files	Notes
`ibdata*`	The InnoDB system tablespace, containing multiple InnoDB tables and associated indexes.	Because the original files might change while the backup is in progress, the `apply-log` step applies the same changes to the corresponding backup files.
`*.ibd`	InnoDB file-per-table tablespaces, each containing a single InnoDB table and associated indexes.	Used for tables created under the `innodb_file_per_table`. Because the original files might change while the backup is in progress, the apply-log step applies the same changes to the corresponding backup files.

File Name, Pattern, or Extension	Relation to Original Data Files	Notes
`*.ibz`	Compressed form of InnoDB data files from the MySQL data directory.	Produced instead of `.ibd` files in a compressed backup. The `ibdata*` files representing the InnoDB system tablespace also receive this extension in a compressed backup. The `.ibz` files are uncompressed for the apply-log step.
`*.frm`	Hold metadata about all MySQL tables.	The database is put into a read-only state while these files are copied. These files are copied without changes.
`*.MYD`	MyISAM table data.	The database is put into a read-only state while these files are copied. These files are copied without changes.
`*.MYI`	MyISAM index data.	The database is put into a read-only state while these files are copied. These files are copied without changes.
`*.CSM`	Metadata for CSV tables.	These files are copied without changes. The `backup_history` and `backup_progress` tables created by `mysqlbackup` use the CSV format, so the backup always includes some files with this extension.
`*.CSV`	Data for CSV tables.	These files are copied without changes. The `backup_history` and `backup_progress` tables created by `mysqlbackup` use the CSV format, so the backup always includes some files with this extension.
`*.MRG`	MERGE storage engine references to other tables.	The database is put into a read-only state while these files are copied. These files are copied without changes.
`*.TRG`	Trigger parameters.	The database is put into a read-only state while these files are copied. These files are copied without changes.
`*.TRN`	Trigger namespace information.	The database is put into a read-only state while these files are copied. These files are copied without changes.
`*.opt`	Database configuration information.	The database is put into a read-only state while these files are copied. These files are copied without changes.

File Name, Pattern, or Extension	Relation to Original Data Files	Notes
`*.par`	Definitions for partitioned tables.	The database is put into a read-only state while these files are copied. These files are copied without changes.
`*.ARM`	Archive storage engine metadata.	The database is put into a read-only state while these files are copied. These files are copied without changes.
`*.ARZ`	Archive storage engine data.	The database is put into a read-only state while these files are copied. These files are copied without changes.
`backup-my.cnf`	Records the configuration parameters that specify the layout of the MySQL data files.	Used in restore operations to reproduce the same layout as when the backup was taken.
`ibbackup_logfile`	A condensed version of the `ib_logfile*` files from the MySQL data directory.	The InnoDB log files (`ib_logfile*`) are fixed-size files that are continuously updated during database operation. For backup purposes, only the changes that are committed while the backup is in progress are needed. These changes are recorded in `ibbackup_logfile`, and used to re-create the `ib_logfile*` files during the apply-log phase.
`ibbackup_redo_log_only`	Used instead of `ibbackup_logfile` for incremental backups taken with the `--incremental-with-redo-log-only` option.	
`ib_logfile*`	Created in the backup directory during the apply-log phase after the initial backup.	These files are not copied from the original data directory, but rather re-created in the backup directory during the apply-log phase after the initial backup, using the changes recorded in the `ibbackup_logfile` file.
`*.bl`	Renamed version of each .isl file from the backed-up server.	A `.isl` file is created when you specify the location of an `InnoDB` table using the syntax `CREATE TABLE ... DATA DIRECTORY = ...`, to act like a symbolic link pointing to the tablespace file. (See Specifying the Location of a Tablespace for details.) The `.bl` files might or might not be turned back into `.isl` files during the `copy-back` operation. If the specified directory does not exist on the

File Name, Pattern, or Extension	Relation to Original Data Files	Notes
		server where the backup is restored, the `mysqlbackup` command attempts to create it. If the directory cannot be created, the restore operation fails. Thus, if you use the `DATA DIRECTORY` clause to put tables in different locations, and restore to a server with a different file structure where the corresponding directories cannot be created, edit the `.bl` files before restoring to point to directories that do exist on the destination server.
Timestamped directory, such as `2011-05-26_13-42-02`	Created by the `--with-timestamp` option. All the backup files go inside this subdirectory.	Use the `--with-timestamp` option whenever you intend to keep more than one set of backup data available under the same main backup directory.
`datadir` directory	A subdirectory that stores all the data files and database subdirectories from the original MySQL instance.	Created under the backup directory by the `mysqlbackup` command.
`image file`	A single-file backup produced by the `backup-to-image` option, with a name specified by the `--backup-image` option.	If your backup data directory consists only of zero-byte files, with a single giant data file in the top-level directory, you have a single-file backup. You can move the image file without losing or damaging the contents inside it, then unpack it with the `mysqlbackup` command using the `extract` option and specifying the same image name with the `--backup-image` option. Although some extra files such as `backup-my.cnf` and the `meta` subdirectory are present in the backup directory, these files are also included in the image file and do not need to be moved along with it.
`any other files`	Copied from the MySQL data directory.	By default, any unrecognized files in the MySQL data directory are copied to the backup. To omit such files, specify the `--only-known-file-types` option.
`meta` directory	A subdirectory that stores files with metadata about the backup.	Created under the backup directory by the `mysqlbackup` command. All files listed below go inside the `meta` subdirectory.

File Name, Pattern, or Extension	Relation to Original Data Files	Notes
`backup_variables.txt`	Holds important information about the backup. For use by the `mysqlbackup` command only. Prior to MySQL Enterprise Backup 3.6, this information was in a file named `ibbackup_binlog_info`.	The `mysqlbackup` command consults and possibly updates this file during operations after the initial backup, such as the apply-log phase or the restore phase.
`image_files.xml`	Contains the list of all the files (except itself) that are present in the single-file backup produced by the `backup-to-image` or `backup-dir-to-image` options. For details about this file, see Section 9.5, "Using the MySQL Enterprise Backup Manifest".	This file is not modified at any stage once generated.
`backup_create.xml`	Lists the command line arguments and environment in which the backup was created. For details about this file, see Section 9.5, "Using the MySQL Enterprise Backup Manifest".	This file is not modified once it is created. You can prevent this file from being generated by specifying the `--disable-manifest` option.
`backup_content.xml`	Essential metadata for the files and database definitions of the backup data. For details about this file, see Section 9.5, "Using the MySQL Enterprise Backup Manifest".	This file is not modified once created. You can prevent this file from being generated by specifying the `--disable-manifest` option.
`comments.txt`	Produced by the `--comments` or `--comments-file` option.	The comments are specified by you to document the purpose or special considerations for this backup job.
`gtid_executed.sql`	Signifies the backup came from a server with GTIDs enabled.	GTIDs are a replication feature in MySQL 5.6 and higher. See Replication with Global Transaction Identifiers for details. When you back up a server with GTIDs enabled, the file `gtid_executed.sql` is created in the backup directory. Edit and execute this file after restoring the backup data on a slave server.

InnoDB Data

Data managed by the InnoDB storage engine is always backed up. The primary InnoDB-related data files that are backed up include the ibdata* files that represent the system tablespace and possibly the data for some user tables; any .ibd files, containing data from user tables created with the file-per-table setting enabled; data extracted from the ib_logfile* files (the redo log information representing changes that occur while the backup is running), which is stored in a new backup file ibbackup_logfile.

If you use the compressed backup feature, the `.ibd` files are renamed in their compressed form to .ibz files.

The files, as they are originally copied, form a raw backup that requires further processing before it is ready to be restored. You then run the apply step, which updates the backup files based on the changes recorded in the `ibbackup_logfile` file, producing a prepared backup. At this point, the backup data corresponds to a single point in time. The files are now ready to be restored to their original location, or for some other use, such as testing, reporting, or deployment as a replication slave.

To restore InnoDB tables to their original state, you must also have the corresponding .frm files along with the backup data. Otherwise, the table definitions could be missing or outdated, if someone has run `ALTER TABLE` or `DROP TABLE` statements since the backup. By default, the `mysqlbackup` command automatically copies the `.frm` files during a backup operation and restores the files during a restore operation.

Data from MyISAM and Other Storage Engines

The `mysqlbackup` command can also back up the .MYD files, .MYI files, and associated .frm files for MyISAM tables. The same applies to files with other extensions, as shown in this list.

> **Note**
>
> While MySQL Enterprise Backup can back up non-InnoDB data (like MYISAM tables), the MySQL server to be backed up must support InnoDB (i.e., the backup process will fail if the server was started up with the `--innodb=OFF` or `--skip-innodb` option), and the server must contain at least one InnoDB table.

MyISAM tables and these other types of files cannot be backed up in the same non-blocking way as InnoDB tables can. This phase is a warm backup: changes to these tables are prevented while they are being backed up, possibly making the database unresponsive for a time, but no shutdown is required during the backup.

> **Note**
>
> To avoid concurrency issues during backups of busy databases, you can use the `--only-innodb` or `--only-innodb-with-frm` option to back up only InnoDB tables and associated data.

Generated Files Included in the Backup

The backup data includes some new files that are produced during the backup process. These files are used to control later tasks such as verifying and restoring the backup data. The files generated during the backup process include:

- `backup-my.cnf`: Records the crucial configuration parameters that apply to the backup. These parameter values are used during a restore operation, so that the original values are used regardless of changes to your `my.cnf` file in the meantime.
- `meta/backup_create.xml`: Lists the command line arguments and environment in which the backup was created.
- `meta/backup_content.xml`: Essential metadata for the files and database definitions of the backup data.

For details about all the files in the backup directory, see Table 1.1, “Files in a MySQL Enterprise Backup Output Directory”.

Single-File Backups

Depending on your workflow, you might perform a single-file backup rather than the typical backup that produces a separate file for every file in the original instance. The single-file format is easier to transfer to a different system, compress and uncompress, and ensure that no backed-up files are deleted later

by mistake. It is just as fast as a multi-file backup to do a full restore; restoring individual files can be slower than in a multi-file backup. For instructions, see Section 3.3.5, “Making a Single-File Backup”.

1.5 Overview of Restoring a Database

To initiate the restore process, you run the `mysqlbackup` command with the `copy-back` subcommand. The MySQL server must be shut down during the restore process. You can restore all the data for a MySQL server -- multiple databases, each containing multiple tables. Or, you can restore selected databases, tables, or both.

To repair a problem such as data corruption, you restore the data back to its original location on the original server machine. You might restore to a different server machine or a different location to set up a new replication slave with the data from a master server, or to clone a database for reporting purposes.

See Chapter 5, *Recovering or Restoring a Database* for instructions on restore techniques.

Chapter 2 Installing MySQL Enterprise Backup

Install the MySQL Enterprise Backup product on each database server whose contents you intend to back up. You perform all backup and restore operations locally, by running the `mysqlbackup` command on the same server as the MySQL instance.

Optional: You can also install the MySQL Enterprise Backup product on computers other than the database server, only to run `mysqlbackup` with the `apply-log` option. See Section 4.1.1.2, “Apply-Log Operations for Existing Backup Data” for information about bringing backup data to a separate server and running the “apply log” step there.

The MySQL Enterprise Backup product is packaged as either an archive file (`.tgz`, archived with `tar` and compressed with `gzip`), or as a platform-specific installer that is more automated and convenient than with the former InnoDB Hot Backup product.

Installing on Unix and Linux Systems

For all Linux and Unix systems, the product is available as a `.tgz` file. Unpack this file as follows:

```
tar xvzf package.tgz
```

The `mysqlbackup` command is unpacked into a subdirectory. You can either copy them into a system directory (preserving their execute permission bits), or add to your `$PATH` setting the directory where you unpacked it.

For certain Linux distributions, the product is also available as an RPM archive. When you install the RPM, using the command `sudo rpm -i package_name.rpm`, the `mysqlbackup` command is installed in the directory `/opt/mysql/meb-3.8`. You must add this directory to your `$PATH` setting.

Installing on Windows Systems

Specify the installation location, preferably relative to the directory where the MySQL Server product is installed.

Choose the option to add this directory to the windows `%PATH%` setting, so that you can run the `mysqlbackup` command from a command prompt.

Verify the installation by selecting the menu item Start > Programs > MySQL Enterprise Backup 3.8.2 > MySQL Enterprise Backup Command Line. This menu item opens a command prompt and runs the `mysqlbackup` command to display its help message showing the option syntax.

`mysqlbackup` Syntax Changes in MySQL Enterprise Backup 3.6 and Higher

In MySQL Enterprise Backup 3.6 and higher, the `mysqlbackup` command takes over the functions formerly performed by the `ibbackup` and `innobackup` commands. As a result, option syntax has changed, and you might need to modify backup scripts to use the new options and remove references to the `ibbackup` command. The new options enable more features and flexibility, and are more consistent with the options used by the `mysqld` client. For the latest syntax information, see Chapter 4, *`mysqlbackup` Command Reference*. For differences between `mysqlbackup` syntax and `ibbackup`/`innobackup` syntax, see Section B.6, “New and Changed Options in MySQL Enterprise Backup 3.6 and Higher”. For how to use the former `ibbackup` and `innobackup` commands during a transition period, see Section B.8, “`ibbackup` and `innobackup` Commands”.

Part II Using MySQL Enterprise Backup

Table of Contents

Chapter 3 Backing Up a Database Server

Table of Contents

This section describes the different kinds of backups that MySQL Enterprise Backup can create and the techniques for producing them, with examples showing the relevant syntax for the `mysqlbackup` command. It also includes a full syntax reference for the `mysqlbackup` command.

3.1 Before the First Backup

The best practices for backups involve planning and strategies. This section outlines some of the preparation needed to put such plans and strategies in place.

3.1.1 Collect Database Information

Before backing up a particular database server for the first time, gather some information and decide on some directory names, as outlined in the following table.

Table 3.1 Information Needed to Back Up a Database

Information to Gather	Where to Find It	How Used
Path to MySQL configuration file	Default system locations, hardcoded application default locations, or from `--defaults-file` option in `mysqld` startup script.	This is the preferred way to convey database configuration information to the `mysqlbackup` command, using the `--defaults-file` option. When connection and data layout information is available from the configuration file, you can skip most of the other choices listed below.
MySQL port	MySQL configuration file or `mysqld` startup script.	Used to connect to the database instance during backup operations. Specified via the `--port` option of `mysqlbackup`. `--port` is not needed if available from MySQL configuration file. Not needed when doing an offline (cold) backup, which works directly

Information to Gather	Where to Find It	How Used
		on the files using OS-level file permissions.
Path to MySQL data directory	MySQL configuration file or `mysqld` startup script.	Used to retrieve files from the database instance during backup operations, and to copy files back to the database instance during restore operations. Automatically retrieved from database connection for hot and warm backups. Taken from MySQL configuration file for cold backups.
ID and password of privileged MySQL user	You record this during installation of your own databases, or get it from the DBA when backing up databases you do not own. Not needed when doing an offline (cold) backup, which works directly on the files using OS-level file permissions. For cold backups, you log in as an administrative user.	Specified via the `--password` option of the `mysqlbackup`. Prompted from the terminal if the `--password` option is present without the password argument.
Path under which to store backup data	You choose this. See Section 3.1.3, "Designate a Location for Backup Data" for details.	By default, this directory must be empty for `mysqlbackup` to write data into it, to avoid overwriting old backups or mixing up data from different backups. Use the `--with-timestamp` option to automatically create a subdirectory with a unique name, when storing multiple sets of backup data under the same main directory.
Owner and permission information for backed-up files (for Linux, Unix, and OS X systems)	In the MySQL data directory.	If you do the backup using a different OS user ID or a different `umask` setting than applies to the original files, you might need to run commands such as `chown` and `chmod` on the backup data. See Section A.1, "Limitations of `mysqlbackup` Command" for details.
Size of InnoDB redo log files	Calculated from the values of the `innodb_log_file_size` and `innodb_log_files_in_group` configuration variables. Use the technique explained for the `--incremental-with-redo-log-only` option.	Only needed if you perform incremental backups using the `--incremental-with-redo-log-only` option rather than the `--incremental` option. The size of the InnoDB redo log and the rate of generation for redo data dictate how often you must perform incremental backups.
Rate at which redo data is generated	Calculated from the values of the InnoDB logical sequence number at different points in time. Use	Only needed if you perform incremental backups using the `--incremental-with-redo-`

Information to Gather	Where to Find It	How Used
	the technique explained for the `--incremental-with-redo-log-only` option.	`log-only` option rather than the `--incremental` option. The size of the InnoDB redo log and the rate of generation for redo data dictate how often you must perform incremental backups.

3.1.2 Grant MySQL Privileges to Backup Administrator

For most backup operations, the `mysqlbackup` command connects to the MySQL server through `--user` and `--password` options. This user requires certain privileges. You can either create a new user with a minimal set of privileges, or use an administrative account such as the root user.

The minimum privileges for the MySQL user that `mysqlbackup` connects are:

- `RELOAD` on all databases and tables.
- `CREATE`, `INSERT`, `DROP`, and `UPDATE` on the tables `mysql.ibbackup_binlog_marker`, `mysql.backup_progress`, and `mysql.backup_history`, and also `SELECT` on `mysql.backup_history`.
- `SUPER`, used to optimize locking and minimize disruption to database processing. This privilege is only needed to back up MySQL 5.5 and higher servers.
- `CREATE TEMPORARY TABLES` for the `mysql` database. This privilege is only needed to back up MySQL 5.5 and higher servers.
- `REPLICATION CLIENT`, to retrieve the binlog position, which is stored with the backup.

To set these privileges for a MySQL user (`dba` in this example) connecting from localhost, issue statements like the following from the `mysql` client program:

```
$ mysql -u root

mysql> GRANT RELOAD ON *.* TO 'dba'@'localhost';

mysql> GRANT CREATE, INSERT, DROP, UPDATE ON mysql.ibbackup_binlog_marker TO 'dba'@'localhost';

mysql> GRANT CREATE, INSERT, DROP, UPDATE ON mysql.backup_progress TO 'dba'@'localhost';

mysql> GRANT CREATE, INSERT, SELECT, DROP, UPDATE ON mysql.backup_history TO 'dba'@'localhost';

mysql> GRANT REPLICATION CLIENT ON *.* TO 'dba'@'localhost';

mysql> GRANT SUPER ON *.* TO 'dba'@'localhost';

mysql> GRANT CREATE TEMPORARY TABLES ON mysql.* TO 'dba'@'localhost';

mysql> FLUSH PRIVILEGES;
```

3.1.3 Designate a Location for Backup Data

All backup-related operations either create new files or reference existing files underneath a specified directory that holds backup data. Choose this directory in advance, on a file system with sufficient storage. (It could even be remotely mounted from a different server.) You specify the path to this directory with the `--backup-dir` option for many invocations of the `mysqlbackup` command.

Once you establish a regular backup schedule with automated jobs, it is preferable to keep each backup within a timestamped subdirectory underneath the main backup directory. To make the `mysqlbackup` command create these subdirectories automatically, specify the `--with-timestamp` option each time you run `mysqlbackup`.

For one-time backup operations, for example when cloning a database to set up a replication slave, you might specify a new directory each time, or specify the `--force` [45] option of `mysqlbackup` to overwrite older backup files.

3.2 The Typical Backup / Verify / Restore Cycle

To illustrate the basic steps in making and using a backup, the following examples show how to do a full backup, examine the data files in the backup directory, and then restore the backup to correct an issue with corruption or lost data.

3.2.1 Backing Up an Entire MySQL Instance

In this example, we specify all required options on the command line for illustration purposes. After testing and standardizing the backup procedure, we could move some options to the MySQL configuration file. The options specify connection information for the database and the location to store the backup data. The final option `backup` specifies the type of operation, because `mysqlbackup` can perform several kinds of backup, restore, and pack/unpack operations.

For this example, we specify the final option as `backup-and-apply-log`. This option performs an extra stage after the initial backup, to bring all InnoDB tables up-to-date with any changes that occurred during the backup operation, so that the backup is immediately ready to be restored. For backups of huge or busy databases, you might split up these stages to minimize load on the database server. That is, run `mysqlbackup` first with the `backup` option, transfer the backup to another server, then run `mysqlbackup` with the `apply-log` option to perform the final processing.

The output echoes all the parameters used by the backup operation, including several that are retrieved automatically using the database connection. The unique ID for this backup job is recorded in special tables that `mysqlbackup` creates inside the instance, allowing you to monitor long-running backups and view the results of previous backups. The final output section repeats the location of the backup data and provides the LSN values that you might use when you perform an incremental backup next time over the full backup that is just made.

```
$ mysqlbackup --port=13000 --protocol=tcp --user=root --password \
  --backup-dir=/home/admin/backups backup-and-apply-log

MySQL Enterprise Backup version 3.7.0
Copyright (c) 2003, 2012, Oracle and/or its affiliates. All Rights Reserved.

INFO: Starting with following command line ...
 mysqlbackup --port=13000 --protocol=tcp --user=root --password
        --backup-dir=/home/admin/backups
        backup

...informational messages...

-----------------------------------------------------------------------
                      Server Repository Options:
-----------------------------------------------------------------------
  datadir                          =  /home/mysql/data/
  innodb_data_home_dir             =  /home/mysql/data
  innodb_data_file_path            =  ibdata1:20M;ibdata2:20M:autoextend
  innodb_log_group_home_dir        =  /home/mysql/data
  innodb_log_files_in_group        =  4
  innodb_log_file_size             =  20971520

-----------------------------------------------------------------------
                      Backup Config Options:
-----------------------------------------------------------------------
  datadir                          =  /home/admin/backups/datadir
  innodb_data_home_dir             =  /home/admin/backups/datadir
  innodb_data_file_path            =  ibdata1:20M;ibdata2:20M:autoextend
  innodb_log_group_home_dir        =  /home/admin/backups/datadir
  innodb_log_files_in_group        =  4
  innodb_log_file_size             =  20971520
```

```
 mysqlbackup: INFO: Unique generated backup id for this is 13071379168342780
...output showing backup progress...
110604  0:51:59 mysqlbackup: INFO: Full backup completed!
 mysqlbackup: INFO: Backup created in directory '/home/admin/backups'

------------------------------------------------------------
   Parameters Summary
------------------------------------------------------------
   Start LSN                  : 36864
   End LSN                    : 50335
------------------------------------------------------------

mysqlbackup completed OK!
```

Now the backup subdirectory is created under the `backup-dir` we specified. The directory name for each new backup is formed from the date and the clock time when the backup run was started, in the local time zone. The backup directory contains the backed-up `ibdata` files and `ibbackup_logfile`. Each subdirectory corresponds to a MySQL database, and contains copies of `.frm`, `.MYD`, `.MYI`, and similar files. For an example of the layout of such a backup directory, see Section C.1, “Sample Directory Structure for Full Backup”.

3.2.2 Verifying a Backup

To verify the backup, restore the backup data on a different server and run the MySQL daemon (`mysqld`) on the new data directory. Then you can execute `SHOW` statements to verify the database and table structure, and execute queries to verify the number of rows, latest updates, and so on.

This is the same general technique to use when you intend to put the backup data to some other use. For example, you might set up a replication slave by making a backup of the master server, or turn a backup into a new MySQL instance for running report queries.

> **Note**
>
> Always do verification against restored data, rather than running `mysqld` with `datadir` pointing to the backup directory. The SQL statements you use to verify the data change the underlying logical sequence number, which would interfere with using the backup directory for subsequent incremental backups.

If you did the backup with the `backup-and-apply-log` option as in the previous example, the backup data is fully consistent and ready to verify. If you only ran the first stage by using the `backup` option, run `mysqlbackup` a second time with the `apply-log` option before doing this verification. (Typically, you run this second phase on the other server after transferring the backup data there, to minimize the load on the original database server.)

See Chapter 5, *Recovering or Restoring a Database* for the procedure to restore the database files on a different server.

Running the `mysqld` daemon on the restored data requires a valid configuration file, which you specify with the `--defaults-file` option of the `mysqld` command. You can reuse most of the settings from the original `my.cnf` file, combined with the `backup-my.cnf` file in the backup directory, which contains only the small subset of parameters required by `mysqlbackup`. Create a new configuration file by concatenating those two files into a new one, and use that configuration file on the server where you do the verification. Edit the resulting file to make sure the `datadir` parameter points to the right location on the verification server. directory. Edit the values for port, socket, and so on if you need to use different connection settings on the verification server.

3.2.3 Restoring a Database at its Original Location

To restore a MySQL instance from a backup:

- Shut down the database server using your usual technique, such as the command `mysqladmin shutdown`.

- Make sure the backup data is fully consistent, by either using the `backup-and-apply-log` option to perform the backup, or running `mysqlbackup` with the `apply-log` option after the initial backup.
- Use the `mysqlbackup` command with the `copy-back` option. This operation copies tables, indexes, metadata, and any other required files back to their original locations as defined by the original MySQL configuration file. For the different combinations of options that you can specify as part of this operation, see Section 4.1.1.3, “Restore an Existing Backup”.

```
$ mysqlbackup --defaults-file=path_to_my.cnf \
  --datadir=path_to_data_directory \
  --backup-dir=path_to_backup_directory copy-back
...many lines of output...
mysqlbackup: Finished copying backup files.

101208 16:48:13 mysqlbackup: mysqlbackup completed OK!
```

Now the original database directory is restored from the backup, and you can restart the database server.

3.3 Backup Scenarios and Examples

All of the following tasks and examples make use of the `mysqlbackup` command. For detailed syntax information, see Chapter 4, *`mysqlbackup` Command Reference*.

3.3.1 Making a Full Backup

Most backup strategies start with a complete backup of the MySQL server, from which you can restore all databases and tables. After you do one full backup, you might do incremental backups (which are smaller and faster) for the next several backup jobs. Periodically, you then do another full backup to begin the cycle again.

This section outlines some of the considerations for making this most basic kind of backup. Because a full backup can take longer and produce larger backup files then other kinds of backups, your decisions about speed, capacity, and convenience are especially important for this part of the backup strategy.

For examples showing the commands to make a full backup, see Section 3.2.1, “Backing Up an Entire MySQL Instance”.

Options on Command Line or in Configuration File?

For clarity, the examples in this manual typically show command-line options to demonstrate connection parameters and other information that might be the same for each backup job. For convenience and consistency, you can include these options in the `[mysqlbackup]` section of the MySQL configuration file that you pass to the `mysqlbackup` command; `mysqlbackup` also picks them up from the `[mysqld]` section if they are present. For example, relying on the port information in the configuration file avoids the need to edit your backup scripts if the database instance switches to a different port.

Output in Single Directory or Timestamped Subdirectories?

For convenience, the `--with-timestamp` option creates uniquely named subdirectories under the backup directory, to hold the output from each backup job. This option is not the default, for backward compatibility for users who relied on the behavior of the former `ibbackup` command, which wrote its output to the top-level backup directory. The timestamped subdirectories make it simpler to establish retention periods, for example by removing or archiving backup data past a certain age.

If you do use a single backup directory (that is, if you omit the `--with-timestamp` option), either specify a new unique directory name for each backup job, or specify the `--force` [45] option to overwrite existing backup files.

With the `--incremental-base` option, as part of each incremental backup command, you specify the directory containing the previous backup. To make the directory names predictable, you might prefer to leave out the `--with-timestamp` option and instead generate a sequence of directory names as part of your backup script.

Always Full Backup, or Full Backup plus Incremental Backups?

If your InnoDB data volume is small, or if your database is so busy that a high percentage of data changes between backups, you might run a full backup each time. Typically, you can save time and storage space by running periodic full backups, and in between running several incremental backups, as described in Section 3.3.2, “Making an Incremental Backup”.

Use Compression or Not?

Doing a compressed backup can save considerable space, allowing you to keep more sets of backup data on a single server. The tradeoff is that you need extra storage space (to hold both compressed and uncompressed data) while preparing the backup to be restored, and in an emergency you might find you do not have spare storage space or the time to uncompress a huge backup. For that reason, compression is more practical for data that is not urgently needed, or while the backup is in transit to another server, where it will be uncompressed for the `apply-log` phase.

3.3.2 Making an Incremental Backup

An incremental backup only backs up data that changed since the previous backup. This technique provides additional flexibility in designing a backup strategy and reduces required storage for backups.

Incremental backups are typically smaller and take less time than a full backup, making them a good choice for frequent backup jobs. Taking frequent incremental backups ensures you can always restore the database to the same state as a few hours or days in the past, without as much load or storage overhead on the database server as taking frequent full backups.

> **Note**
>
> Because an incremental backup always adds to an existing set of backup files, make at least one full backup before doing any incremental backups.

Incremental backup is enabled through an option to the `mysqlbackup` command. For straightforward incremental backups, specify the `--incremental` option. An alternative method uses the `--incremental-with-redo-log-only` option, requiring additional planning on your part.

You also indicate the point in time of the previous full or incremental backup. For convenience, you can use the `--incremental-base` option to automatically derive the necessary log sequence number (LSN) from the metadata stored in a previous backup directory. Or, you can specify an explicit LSN value using the `--start-lsn` option, using the ending LSN from a previous full or incremental backup.

To prepare the backup data to be restored, you combine each incremental backup with an original full backup. Typically, you perform a new full backup after a designated period of time, after which you can discard the older incremental backup data.

When running the “apply log” step for an incremental backup, you specify the option sequence `--incremental apply-log`, and the paths to 2 MySQL configuration files, first the `.cnf` file pointing to the full backup that you are updating, then the `.cnf` file pointing to the incremental backup data files. If you have taken several incremental backups since the last full backup, you might run several such “apply log” steps, one after the other, to bring the full backup entirely up to date.

Space Considerations for Incremental Backups

The incremental backup feature is primarily intended for InnoDB tables, or non-InnoDB tables that are read-only or rarely updated. For non-InnoDB files, the entire file is included in an incremental backup if that file changed since the previous backup.

You cannot perform incremental backups with the `--compress` option.

Incremental backups detect changes at the level of pages in the InnoDB data files, as opposed to table rows; each page that has changed is backed up. Thus, the space and time savings are not exactly proportional to the percentage of changed InnoDB rows or columns.

When an InnoDB table is dropped and you do a subsequent incremental backup, the `apply-log` step removes the corresponding `.ibd` file from the full backup directory. Since the backup program cannot have the same insight into the purpose of non-InnoDB files, when a non-InnoDB file is removed between the time of a full backup and a subsequent incremental backup, the apply-log step does not remove that file from the full backup directory. Thus, restoring a backup could result in a deleted file reappearing.

Examples of Incremental Backups

This example uses the `mysqlbackup` command to make an incremental backup of a MySQL server, including all databases and tables. We show two alternatives, one using the `--incremental-base` option and the other using the `--start-lsn` option.

With the `--incremental-base` option, you do not have to keep track of LSN values between one backup and the next. Instead, you specify the directory of the previous backup (either full or incremental), and `mysqlbackup` figures out the starting point for this backup based on the metadata of the earlier one. Because you need a known set of directory names, you might use hardcoded names or generate a sequence of names in your own backup script, rather than using the `--with-timestamp` option.

```
$ mysqlbackup --defaults-file=/home/pekka/.my.cnf --incremental \
  --incremental-base=dir:/incr-backup/wednesday \
  --incremental-backup-dir=/incr-backup/thursday \
  backup
...many lines of output...
mysqlbackup: Backup created in directory '/incr-backup/thursday'
mysqlbackup: start_lsn: 2654255717
mysqlbackup: incremental_base_lsn: 2666733462
mysqlbackup: end_lsn: 2666736714

101208 17:14:58 mysqlbackup: mysqlbackup completed OK!
```

With the `--start-lsn` option, you do have to record the LSN of the previous backup, but then the location of the previous backup is less significant, so you can use `--with-timestamp` to create named subdirectories automatically.

```
$ mysqlbackup --defaults-file=/home/pekka/.my.cnf --incremental \
  --start-lsn=2654255716 \
  --with-timestamp \
  --incremental-backup-dir=/incr-backup \
  backup
...many lines of output...
mysqlbackup: Backup created in directory '/incr-backup/2010-12-08_17-14-48'
mysqlbackup: start_lsn: 2654255717
mysqlbackup: incremental_base_lsn: 2666733462
mysqlbackup: end_lsn: 2666736714

101208 17:14:58 mysqlbackup: mysqlbackup completed OK!
```

Wherever you use the `--incremental` option, you can use the `--incremental-with-redo-log-only` option instead. Because `--incremental-with-redo-log-only` is more dependent on the precise LSN than the `--incremental` option is, use the `--incremental-base` option rather than the `--start-lsn` option with this kind of incremental backup.

For this alternative kind of incremental backup to work, the volume of changed information must be low enough, and the redo log files must be large enough, that all the changes since the previous

incremental backup must be present in the redo log and not overwritten. See the `--incremental-with-redo-log-only` option description to learn how to verify those requirements.

```
$ mysqlbackup --defaults-file=/home/pekka/.my.cnf --incremental \
  --incremental-base=dir:/incr-backup/wednesday \
  --incremental-backup-dir=/incr-backup/thursday \
  backup
...many lines of output...
mysqlbackup: Backup created in directory '/incr-backup/thursday'
mysqlbackup: start_lsn: 2654255717
mysqlbackup: incremental_base_lsn: 2666733462
mysqlbackup: end_lsn: 2666736714

101208 17:14:58 mysqlbackup: mysqlbackup completed OK!
```

See Section C.3, “Sample Directory Structure for Incremental Backup” for a listing of files from a typical incremental backup.

Once again, we apply to the full backup any changes that occurred while the backup was running:

```
$ mysqlbackup --backup-dir=/full-backup/2010-12-08_17-14-11 apply-log
..many lines of output...
101208 17:15:10  mysqlbackup: Full backup prepared for recovery successfully!

101208 17:15:10 mysqlbackup: mysqlbackup completed OK!
```

Then, we apply the changes from the incremental backup:

```
$ mysqlbackup --incremental-backup-dir=/incr-backup/2010-12-08_17-14-48
  --backup-dir=/full-backup/2010-12-08_17-14-11 apply-incremental-backup
...many lines of output...
101208 17:15:12 mysqlbackup: mysqlbackup completed OK!
```

Now, the data files in the full backup directory are fully up-to-date, as of the time of the last incremental backup.

This example shows an incremental backup. The last full backup we ran reported that the highest LSN was 2638548215:

```
mysqlbackup: Was able to parse the log up to lsn 2638548215
```

We specify that number again in the command here; the incremental backup includes all changes that came *after* the specified LSN.

```
$ mysqlbackup --defaults-file=/home/pekka/.my.cnf --incremental \
  --start-lsn=2638548215 \
  --incremental-backup-dir=/incr-backup/2010-12-08_17-14-48 \
  --backup-dir=/full-backup/2010-12-08_17-14-11 \
 backup
...many lines of output...
mysqlbackup: Scanned log up to lsn 2654252454.
mysqlbackup: Was able to parse the log up to lsn 2654252454.
mysqlbackup: Maximum page number for a log record 0
mysqlbackup: Backup contains changes from lsn 2638548216 to lsn 2654252454
101208 17:12:24  mysqlbackup: Incremental backup completed!
```

Next steps:

- Make a note of the LSN value in the message at the end of the backup, for example, `mysqlbackup: Was able to parse the log up to lsn` *`LSN_number`*. You specify this value when performing incremental backups of changes that occur after this incremental backup.

- Apply the incremental backup to the backup files, so that the backup is ready to be restored at any time. You can move the backup data to a different server first, to avoid the CPU and I/O overhead of this operation on the database server itself.
- On a regular schedule, determined by date or amount of database activity, take further take incremental backups.
- Optionally, periodically start the cycle over again by taking a full uncompressed or compressed backup. Typically, this milestone happens when you can archive and clear out your oldest backup data.

3.3.3 Making a Compressed Backup

To save disk space, you can compress InnoDB backup data files by using the `--compress` option of `mysqlbackup`. Compression lets you keep more sets of backup data on hand, and saves on transmission time when sending the backup data to another server. The downside is extra CPU overhead during the backup itself, and extra time needed during the restore process as the data is uncompressed.

The backup compression feature only applies to InnoDB tables. MySQL 5.5 and higher make InnoDB the default storage engine, because of its high concurrency, reliability, and fast crash recovery. The hot backup and incremental backup features of MySQL Enterprise Backup also apply only to InnoDB tables; For these reasons, Oracle recommends using InnoDB tables for your biggest, busiest, and most important data.

When InnoDB tablespace files are compressed during backup, they receive the extension `.ibz` rather than the usual `.ibd` extension. To avoid wasting CPU cycles without saving additional disk space, `--compress` does not attempt to compress already-compressed tables that use the Barracuda file format; such tablespace files keep the usual `.ibd` extension.

> **Note**
>
> If there is unused space within an InnoDB tablespace file, the entire file is copied during an uncompressed backup. Do a compressed backup to avoid the storage overhead for this unused space.

You can only use the `--compress` option for full backups, not for incremental backups.

```
$ mysqlbackup --defaults-file=/home/pekka/my.cnf --compress backup
...many lines of output...
mysqlbackup: Compressed 488 MB of data files to 53 MB (compression 89%).

101208 15:48:09  mysqlbackup: Full backup completed!
```

The backup directory is shown below. Compressed data files have the suffix `.ibz`. Typically, compression ratios of more than 70% are achieved:

```
$ ls -l /sqldata-backup
total 54676
-rw-r--r-- 1 pekka pekka      158 2010-12-08 15:48 ibbackup_export_variables.txt
-rw-r----- 1 pekka pekka     1024 2010-12-08 15:48 ibbackup_logfile
-rw-r----- 1 pekka pekka  1095854 2010-12-08 15:47 ibdata1.ibz
-rw-r----- 1 pekka pekka   811625 2010-12-08 15:47 ibdata2.ibz
-rw-r----- 1 pekka pekka 54058462 2010-12-08 15:48 ibdata3.ibz
```

Next steps:

- Make a note of the LSN value in the message at the end of both full and incremental backups, for example, `mysqlbackup: Was able to parse the log up to lsn` *`LSN_number`*. You specify this value when performing incremental backups of changes that occur after this full backup.

- Apply the log to the compressed backup files, so that the full backup is ready to be restored at any time. You can move the backup data to a different server first, to avoid the CPU and I/O overhead of performing this operation on the database server.
- After applying the log, periodically take incremental backups, which are much faster and smaller than a full backup like this.

3.3.4 Making a Partial Backup

By default, all the files in the data directory are included in the backup, so the backup includes data from all MySQL storage engines, any third-party storage engines, and even any non-database files in that directory. This section explains options you can use to selectively back up or exclude data.

MySQL Enterprise Backup can make several kinds of partial backup:

- Leaving out files that are present in the MySQL data directory but not actually part of the MySQL instance. This operation involves the `--only-known-file-types` option.
- Including certain InnoDB tables but not others. This operation involves the `--include`, `--only-innodb`, and `--only-innodb-with-frm` options.
- Including certain non-InnoDB tables from selected databases but not others. This operation involves the `--databases` and `--databases-list-file` options.

For syntax details on all these options, see Section 4.1.9, “Partial Backup Options”.

> **Note**
>
> Typically, a partial backup is more difficult to restore than a full backup, because the backup data might not include the necessary interrelated pieces to constitute a complete MySQL instance. In particular, InnoDB tables have internal IDs and other data values that can only be restored to the same instance, not a different MySQL server. Always fully test the recovery procedure for any partial backups to understand the relevant procedures and restrictions.

3.3.4.1 Backing Up Some or All InnoDB Tables

With its `--include` option, `mysqlbackup` can make a backup that includes some InnoDB tables but not others:

- A partial backup with the `--include` option always contains the InnoDB system tablespace and all the tables inside it.
- For the InnoDB tables stored outside the system tablespace, the partial backup includes only those tables whose names match the regular expression specified with the `--include` option.

This operation requires the tables being left out to be stored in separate *`table_name`*`.ibd` files. To put an InnoDB table outside the system tablespace, create it while the `innodb_file_per_table` MySQL configuration option is enabled. Each `.ibd` file holds the data and indexes of one table only.

Those InnoDB tables created with `innodb_file_per_table` turned off are stored as usual in the InnoDB system tablespace, and cannot be left out of the backup.

For each table with a per-table data file a string of the form `db_name.table_name` is checked against the regular expression specified with the `--include` option. If the regular expression matches the complete string `db_name.table_name`, the table is included in the backup. The regular expression uses the POSIXextended form. On Unix-like systems, quote the regular expression appropriately to prevent interpretation of shell meta-characters. This feature has been implemented with Henry Spencer's regular expression library.

IMPORTANT: Although the `mysqlbackup` command supports taking partial backups, be careful when restoring a database from a partial backup. `mysqlbackup` copies also the `.frm` files of those

tables that are not included in the backup, except when you do partial backups using, for example, the `--databases` option. If you use `mysqlbackup` with the `--include` option, before restoring the database, delete from the backup data the `.frm` files for any tables that are not included in the backup.

IMPORTANT: Because the InnoDB system tablespace holds metadata about InnoDB tables from all databases in an instance, restoring a partial backup on a server that includes other databases could cause the system to lose track of those InnoDB tables in other databases. Always restore partial backups on a fresh MySQL server instance without any other InnoDB tables that you want to preserve.

The `--only-innodb` and `--only-innodb-with-frm` options back up InnoDB tables only, skipping those of other storage engines. You might use one of these options for some backup operations based on the following considerations:

- The InnoDB tables are backed up using the hot backup technique, which does not interfere with database processing.
- The `--compress`, `--incremental`, and `--incremental-with-redo-log-only` options offer benefits only for InnoDB data.
- In a busy production environment, InnoDB tables might represent the bulk of your important data because of the importance of high concurrency and crash recovery.
- In MySQL 5.5 and higher, InnoDB is the default storage engine for new tables.

Example 3.1 Making an Uncompressed Backup of InnoDB Tables

In this example, the options file `/home/pekka/.my.cnf` defines the MySQL installation to back up. Running `mysqlbackup` performs the first phase of the process:

```
# Back up all InnoDB tables but no .frm files.
$ mysqlbackup --defaults-file=/home/pekka/.my.cnf --only-innodb backup
...many lines of output...
mysqlbackup: Scanned log up to lsn 32164666892.
mysqlbackup: Was able to parse the log up to lsn 32164666892.
mysqlbackup: Maximum page number for a log record 0
101208 15:33:11  mysqlbackup: Full backup completed!

# Back up all InnoDB tables and corresponding .frm files.
$ mysqlbackup --defaults-file=/home/pekka/.my.cnf --only-innodb-with-frm backup
...many lines of output...
mysqlbackup: Scanned log up to lsn 32164666892.
mysqlbackup: Was able to parse the log up to lsn 32164666892.
mysqlbackup: Maximum page number for a log record 0
101208 15:33:11  mysqlbackup: Full backup completed!
```

The backup directory now contains a backup log file and copies of InnoDB data files. The backup directory from the `--only-innodb-with-frm` option also includes `.frm` files for the InnoDB tables.

Next Steps:

- Make a note of the LSN value in the message at the end of both full and incremental backups, for example, `mysqlbackup: Was able to parse the log up to lsn` *`LSN_number`*. You specify this value when performing incremental backups of changes that occur after this full backup.
- Apply the log to the uncompressed backup files, so that the full backup is ready to be restored at any time. You can move the backup data to a different server first, to avoid the CPU and I/O overhead of performing this operation on the database server.
- After applying the log, periodically take incremental backups, which are much faster and smaller than a full backup like this.

Example 3.2 Making an Uncompressed Partial Backup of InnoDB Tables

In this example, we have configured MySQL so that some InnoDB tables have their own tablespaces. We make a partial backup including only those InnoDB tables in `test` database whose name starts with `ib`. The contents of the database directory for `test` database are shown below. The directory contains a MySQL description file (`.frm` file) for each of the tables (`alex1`, `alex2`, `alex3`, `blobt3`, `ibstest0`, `ibstest09`, `ibtest11a`, `ibtest11b`, `ibtest11c`, and `ibtest11d`) in the database. Of these 10 tables six (`alex1`, `alex2`, `alex3`, `blobt3`, `ibstest0`, `ibstest09`) are stored in per-table datafiles (`.ibd` files).

```
$ ls /sqldata/mts/test
alex1.frm  alex2.ibd  blobt3.frm    ibstest0.ibd  ibtest11a.frm  ibtest11d.frm
alex1.ibd  alex3.frm  blobt3.ibd    ibtest09.frm  ibtest11b.frm
alex2.frm  alex3.ibd  ibstest0.frm  ibtest09.ibd  ibtest11c.frm
```

We run the `mysqlbackup` with the `--include` option:

```
# Back up some InnoDB tables but not any .frm files.
$ mysqlbackup --defaults-file=/home/pekka/.my.cnf --include='test\.ib.*' --only-innodb backup
...many lines of output...
mysqlbackup: Scanned log up to lsn 2666737471.
mysqlbackup: Was able to parse the log up to lsn 2666737471.
mysqlbackup: Maximum page number for a log record 0
101208 17:17:45  mysqlbackup: Full backup completed!

# Back up some InnoDB tables and the .frm files for the backed-up tables only.
$ mysqlbackup --defaults-file=/home/pekka/.my.cnf --include='test\.ib.*' \
  --only-innodb-with-frm=related backup
...many lines of output...
mysqlbackup: Scanned log up to lsn 2666737471.
mysqlbackup: Was able to parse the log up to lsn 2666737471.
mysqlbackup: Maximum page number for a log record 0
101208 17:17:45  mysqlbackup: Full backup completed!
```

The backup directory contains only backups of `ibstest` and `ibtest09` tables. Other InnoDB tables did not match the include pattern `test\.ib.*`. Notice, however, that the tables `ibtest11a`, `ibtest11b`, `ibtest11c`, `ibtest11d` are in the backup even though they are not visible in the directory shown below, because they are stored in the system tablespace (`ibdata1` file) which is always included in the backup.

```
# With the --only-innodb option:
$ ls /sqldata-backup/test
ibstest0.ibd   ibtest09.ibd

# With the --only-innodb-with-frm=related option:
$ ls /sqldata-backup/test
ibstest0.frm   ibtest09.frm
ibstest0.ibd   ibtest09.ibd
```

Example 3.3 Making a Compressed Partial Backup

We have configured MySQL so that every InnoDB table has its own tablespace. We make a partial backup including only those InnoDB tables whose name starts with `alex` or `blob`. The contents of the database directory for `test` database is shown below.

```
$ ls /sqldata/mts/test
alex1.frm  alex2.ibd  blobt3.frm    ibstest0.ibd  ibtest11a.frm  ibtest11d.frm
alex1.ibd  alex3.frm  blobt3.ibd    ibtest09.frm  ibtest11b.frm
alex2.frm  alex3.ibd  ibstest0.frm  ibtest09.ibd  ibtest11c.frm
```

We run `mysqlbackup` with the `--compress` and `--include` options:

```
$ mysqlbackup --defaults-file=/home/pekka/.my.cnf --compress \
  --include='.*\.(alex|blob).*' --only-innodb backup
...many lines of output...
mysqlbackup: Scanned log up to lsn 2666737471.
mysqlbackup: Was able to parse the log up to lsn 2666737471.
mysqlbackup: Maximum page number for a log record 0

mysqlbackup: Compressed 147 MB of data files to 15 MB (compression 89%).

101208 17:18:04  mysqlbackup: Full backup completed!
```

The backup directory for the database `test` is shown below. The `.ibz` files are compressed per-table datafiles.

```
$ ls /sqldata-backup/test
alex1.ibz   alex2.ibz   alex3.ibz   blobt3.ibz
```

3.3.4.2 Omitting Unknown Files

The `--only-known-file-types` option of the `mysqlbackup` command limits the backup to only those files that represent known data files from MySQL or its built-in storage engines, such as `.frm`, `.ibd`, `.myd`, and so on. (See the full list of extensions.) By default, the `mysqlbackup` command backs up all file extensions within the data directory, which could include files produced by many different storage engines. Use this option to omit the additional data files from other storage engines from the backup, for performance or space reasons.

3.3.4.3 Backing Up Non-InnoDB Tables from Selected Databases

The `--databases` and `--databases-list-file` options of the `mysqlbackup` command let you back up non-InnoDB tables only from selected databases, rather than across the entire MySQL instance. (To filter InnoDB tables, use the `--include` option instead.) With `--databases`, you specify a space-separated list of database names, with the entire list enclosed in double quotation marks. With `--databases-list-file`, you specify the path of a file containing the list of database names, one per line.

Some or all of the database names can be qualified with table names, to only back up selected non-InnoDB tables from those databases.

If you specify this option, make sure to include the same set of databases for every backup (especially incremental backups), so that you do not restore out-of-date versions of any databases.

3.3.5 Making a Single-File Backup

To avoid a large number of backup files to track and keep safe, and to simplify moving backup data around, the `mysqlbackup` command can create a backup in a single-file format, pack an existing backup into a single file, unpack the single file back to the original backup directory structure, list the contents of a single-file backup, verify the contents of a single-file backup against embedded checksums, or extract a single file or directory tree. For the syntax of the relevant `mysqlbackup` options, see Section 4.1.10, “Single-File Backup Options”.

Because the single-file backup can be streamed or piped to another process, such as a tape backup or a command such as `scp`, you can use this technique to put the backup on another storage device or server without significant storage overhead on the original database server. (During preparation of the single-file backup, some small work files are prepared temporarily inside the specified backup directory.)

To create a single-file backup, specify the `mysqlbackup` option `backup-to-image`. All the original data files must be under a single directory, rather than spread across different paths. Specify the same path for the `datadir`, `innodb_log_group_home_dir`, and `innodb_data_home_dir` configuration options.

Example 3.4 Single-File Backup to Absolute Path

This command creates a single backup image in the given absolute path. It still requires `--backup-dir`, which is used to hold temporary output, status, and metadata files.

```
mysqlbackup --backup-image=/backups/sales.mbi --backup-dir=/backup-tmp backup-to-image
```

Example 3.5 Single-File Backup to Relative Path

This command specifies `--backup-image` with a relative path underneath the backup directory. The resulting single-file backup is created as `/backups/sales.mbi`.

```
mysqlbackup --backup-image=sales.mbi --backup-dir=/backups backup-to-image
```

Example 3.6 Single-File Backup to Standard Output

The following command dumps the backup output to standard output. Again, the `--backup-dir` directory specified in `my.cnf` is used as a temporary directory.

```
mysqlbackup --backup-dir=/backups --backup-image=- backup-to-image > /backup/mybackup.mbi
```

Example 3.7 Convert Existing Backup Directory to Single Image

The `backup-dir` directory specified in `my.cnf` is bundled into the `/backup/my.mbi` file. The directory can contain anything, not necessarily a backup produced by MySQL Enterprise Backup.

```
mysqlbackup --backup-image=/backup/my.mbi --backup-dir=/var/mysql/backup backup-dir-to-image
```

Example 3.8 Extract Existing Image to Backup Directory

The image contents are unpacked into *`backup-dir`*.

```
mysqlbackup --backup-dir=/var/backup --backup-image=/backup/my.mbi image-to-backup-dir
```

Example 3.9 List Single-File Backup Contents

The image contents are listed with each line indicating a file or directory entry.

```
mysqlbackup --backup-image=/backup/my.mbi list-image
```

Example 3.10 Extract Single-File Backup into Current Directory

The following command extracts all contents from a single-file backup into the current working directory.

```
mysqlbackup --backup-image=/var/my.mbi extract
```

Example 3.11 Extract Single-File Backup into a Backup Directory

This command behaves like the `image-to-backup-dir` option, by extracting all contents of a single-file backup into the `--backup-dir` directory.

```
mysqlbackup --backup-image=/var/my.mbi --backup-dir=/var/backup extract
```

Example 3.12 Selective Extract of Single File

The following command extracts the single file `meta/comments.txt` into the local path `./meta/comments.txt`.

```
mysqlbackup --backup-image=/var/my.mbi \
  --src-entry=meta/comments.txt extract
```

The following command extracts the `meta/comments.txt` file into a specified path `/tmp/mycomments.txt` by using the `--dst-entry` option.

```
mysqlbackup --backup-image=/var/my.mbi \
  --src-entry=meta/comments.txt \
  --dst-entry=/tmp/mycomments.txt extract
```

The following command dumps the contents of `meta/comments.txt` (inside a single-file backup) to standard output.

```
mysqlbackup --backup-image=/var/my.mbi --src-entry=meta/comments.txt --dst-entry=- extract
```

Example 3.13 Selective Extract of Single Directory

The following command extracts a single directory `meta` into a local file system path `./meta`. Extracting a directory extracts all its contents, including any subdirectories.

```
mysqlbackup --backup-image=/backup/my.mbi --src-entry=meta extract
```

The following command extracts all `meta` directory contents (all its files and subdirectories) into the directory `/tmp/my-meta`.

```
mysqlbackup --backup-image=/backup/my.mbi --src-entry=meta \
  --dst-entry=/tmp/my-meta extract
```

Example 3.14 Checking a Single-File Backup for Corruption Problems

For an example showing the output of successful and unsuccesful checks, see Section C.4, “Validating a Single-File Backup Image”.

Example 3.15 Dealing with Absolute Path Names

Since absolute pathnames are extracted to the same paths in local system, it could be a problem if you do not have write permission for that path. You can remap absolute paths as follows:

```
mysqlbackup --backup-image=/backup/my.mbi --src-entry=/ --dst-entry=/myroot extract
mysqlbackup --backup-image=/backup/my.mbi --src-entry=. extract
```

The first command extracts all absolute paths to `/myroot` directory in the local system. The second command extracts all relative paths to the current directory.

3.3.5.1 Streaming the Backup Data to Another Device or Server

To limit the storage overhead on the database server, you can transfer the backup data to a different server without ever storing it locally. The primary MySQL Enterprise Backup feature related to streaming is the single-image backup. To send the single-file backup to standard output, specify by the `mysqlbackup` option `backup-to-image` with no `--backup-image` option. (You can also specify `--backup-image=-` to make it obvious that the data is sent to stdout.) To stream the data, you use the single-file backup in combination with operating system features such as pipes, `ssh/scp`, and so on that can take input from standard output and create an equivalent file on a remote system. You can either store the single-file backup directly on the remote system, or invoke the `mysqlbackup` command with the `image-to-backup-dir` option on the other end to reproduce the directory structure of a regular backup.

Example 3.16 Single-File Backup to a Remote Host

The following command streams the backup output to a remote host, where it is saved directly to a tape device. `--backup-dir=/tmp` designates the directory for storing temporary work files rather than the final output file. For simplicity, all the connection and other necessary options are assumed to be taken from the default configuration file. For the operation to run on the remote system, substitute the combination of command, device, and so on that you use as part of your normal archiving procedure, such as `dd` or `tar`.

```
mysqlbackup --backup-image=- --backup-dir=/tmp backup-to-image | \
ssh user@host command arg1 arg2...
```

3.3.5.2 Backing Up to Tape

Tape drives are affordable, high-capacity storage devices for backup data. The MySQL Enterprise Backup product can interface with media management software (MMS) such as Oracle Secure Backup (OSB) to drive MySQL backup and restore jobs. The media management software must support Version 2 or higher of the System Backup to Tape (SBT) interface.

For information about doing tape backups in combination with MMS products such as Oracle Secure Backup, see Chapter 8, *Using MySQL Enterprise Backup with Media Management Software (MMS) Products*.

3.3.6 Backing Up In-Memory Database Data

The `--exec-when-locked` option of the `mysqlbackup` command lets you specify a command and arguments to run near the end of the backup, while the database is still locked. This command can copy or create additional files in the backup directory. For example, you can use this option to back up `MEMORY` tables with the `mysqldump` command, storing the output in the backup directory. To delay any redirection or variable substitution until the command is executed, enclose the entire parameter value within single quotes.

Chapter 4 `mysqlbackup` Command Reference

Table of Contents

The `mysqlbackup` command is an easy-to-use tool for all backup and restore operations. During backup operations, `mysqlbackup` backs up:

- All InnoDB tables and indexes, including:
  - The InnoDB system tablespace, which by default contains all the InnoDB tables.
  - Any separate data files produced under the InnoDB file-per-table setting. Each one contains one table and its associated indexes. Each data file can use either the original Antelope or the new Barracuda file format.
- All MyISAM tables and indexes.
- Tables managed by other storage engines.
- Other files underneath the MySQL data directory, such as the `.frm` files that record the structure of each table.

In addition to creating backups, `mysqlbackup` can pack and unpack backup data, apply to the backup data any changes to InnoDB tables that occurred during the backup operation, and restore data, index, and log files back to their original locations.

Sample command line arguments to start `mysqlbackup` are:

```
# Information about data files can be retrieved through the database connection.
# Specify connection options on the command line.
mysqlbackup --user=dba --password --port=3306 \
  --with-timestamp --backup-dir=/export/backups \
  backup

# Or we can include the above options in the configuration file
# under [mysqlbackup], and just specify the configuration file
# and the 'backup' operation.
mysqlbackup --defaults-file=/usr/local/mysql/my.cnf backup
```

```
# Or we can specify the configuration file as above, but
# override some of those options on the command line.
mysqlbackup --defaults-file=/usr/local/mysql/my.cnf \
  --compress --user=backupadmin --password --port=18080 \
  backup
```

The `--user` and the `--password` you specify are used to connect to the MySQL server. This MySQL user must have certain privileges in the MySQL server, as described in Section 3.1.2, “Grant MySQL Privileges to Backup Administrator”.

The `--with-timestamp` option places the backup in a subdirectory created under the directory you specified above. The name of the backup subdirectory is formed from the date and the clock time of the backup run.

For the meanings of other command-line options, see Section 4.1, “`mysqlbackup` Command-Line Options”. For information about configuration parameters, see Section 4.2, “Configuration Files and Parameters”.

Make sure that the user or the cron job running `mysqlbackup` has the rights to copy files from the MySQL database directories to the backup directory.

Make sure that your connection timeouts are long enough so that the command can keep the connection to the server open for the duration of the backup run. `mysqlbackup` pings the server after copying each database to keep the connection alive.

IMPORTANT:

- Although the `mysqlbackup` command backs up InnoDB tables without interrupting database use, the final stage that copies non-InnoDB files (such as MyISAM tables and `.frm` files) temporarily puts the database into a read-only state, using the statement `FLUSH TABLES WITH READ LOCK`. For best backup performance and minimal impact on database processing:

  1. Do not run long `SELECT` queries or other SQL statements at the time of the backup run.
  2. Keep your MyISAM tables relatively small and primarily for read-only or read-mostly work.

  Then the locked phase at the end of a `mysqlbackup` run is short (maybe a few seconds), and does not disturb the normal processing of `mysqld` much. If the preceding conditions are not met in your database application, use the `--only-innodb` or `--only-innodb-with-frm` option to back up only InnoDB tables, or use the `--no-locking` option to back up non-InnoDB files. Note that MyISAM, `.frm`, and other files copied under the `--no-locking` setting cannot be guaranteed to be consistent, if they are updated during this final phase of the backup.

- For a large database, a backup run might take a long time. Always check that `mysqlbackup` has completed successfully, either by verifying that the `mysqlbackup` command returned exit code 0, or by observing that `mysqlbackup` has printed the text “mysqlbackup completed OK!”.

- The `mysqlbackup` command is not the same as the former “MySQL Backup” open source project from the MySQL 6.0 source tree. The MySQL Enterprise Backup product supersedes the MySQL Backup initiative.

- Schedule backups during periods when no DDL operations involving tables are running. See Section A.1, “Limitations of `mysqlbackup` Command” for restrictions on backups at the same time as DDL operations.

4.1 mysqlbackup Command-Line Options

The following sections describe the different modes of operation for the mysqlbackup, then explain the applicable options for each mode, and the purpose and operation of each option. For the sets of options that are typically specified together for the various backup and restore tasks, see Section 4.1.1, “Subcommands”.

> **Note**
>
> The mysqlbackup command follows MySQL standard practice for handling duplicate options, whether specified in a configuration file, on the command line, or both. Options are processed first from configuration files, then from the command line. If an option is specified more than once, the last instance takes precedence.

4.1.1 Subcommands

These options represent the major operations or modes for the mysqlbackup command. Only one can be specified for each mysqlbackup invocation, and the name is not preceded by any dashes.

Each of these major options has its own set of required or allowed command parameters. For example, the backup* options require connection information to the database server. The apply-log, and other options that operate on the backup data after it is produced, require options to specify where the backup data is located.

The major groups of subcommands are:

- Backup operations: backup, backup-and-apply-log, backup-to-image
- Apply operations: apply-log, apply-incremental-backup
- Restore operations: copy-back
- Single-file backup operations: image-to-backup-dir, backup-dir-to-image, list-image, extract, validate

4.1.1.1 Backup Operations

The backup operations are the most frequently performed tasks by MySQL Enterprise Backup. Various kinds of backups can be performed by adding different options, like using --compress or --incremental for compressed or incremental backups. Here is the syntax for the mysqlbackup command for performing a backup operation:

```
mysqlbackup [STD-OPTIONS]
            [CONNECTION-OPTIONS]
            [SERVER-REPOSITORY-OPTIONS]
            [BACKUP-REPOSITORY-OPTIONS]
            [METADATA-OPTIONS]
            [COMPRESSION-OPTIONS]
            [SPECIAL-BACKUP-TYPES-OPTIONS]
            [INCREMENTAL-BACKUP-OPTIONS]
            [PARTIAL-BACKUP-OPTIONS]
            [SINGLE-FILE-BACKUP-OPTIONS]
            [PERFORMANCE-SCALABILITY-CAPACITY-OPTIONS]
            [PROGRESS-REPORT-OPTIONS]
            backup | backup-and-apply-log | backup-to-image
```

- backup

  Performs the initial phase of a backup. The second phase is performed later by running mysqlbackup again with the apply-log option.

- `backup-and-apply-log`

  A combination of `backup` and `apply-log`. Not compatible with incremental backups. Any `--compress` option is ignored.

- `backup-to-image`

  Produces a single-file backup rather than a directory structure holding the backup files. Requires the `--backup-image` option to specify the destination file. Can be used to stream the backup to a storage device or another system without ever storing the data on the database server. You can specify `--backup-image=-`, representing standard output, allowing the output to be piped to another command. To avoid mixing normal informational messages with backup output, the `--help` message, errors, alerts, and normal informational messages are always printed to standard error.

Example 4.1 Simple Backup with Connection Parameters from Default Configuration File

The following example shows a minimal backup with the `mysqlbackup` command, with any necessary connection parameters for the database in the `[mysqlbackup]` section of the default MySQL configuration file:

```
mysqlbackup --backup-dir=/export/backups/latest backup
```

Example 4.2 Basic Incremental Backup

```
mysqlbackup --incremental --start-lsn=12345 --incremental-backup-dir=/path/to/incbackup backup
```

There is a separate directory dedicated to incremental backup. Both this directory and the one for full backups can be specified in the `my.cnf` file, and the appropriate directory is used depending on the type of backup. Both the incremental backup data and an earlier full backup are needed to do a successful restore operation.

4.1.1.2 Apply-Log Operations for Existing Backup Data

These operations bring the backup files up-to-date with any changes to InnoDB tables that happened while the backup was in progress. Although for convenience you can combine this operation with the initial backup using the `backup-and-apply-log` option, you must run the steps separately when performing incremental or compressed backups.

```
mysqlbackup [STD-OPTIONS]
            [--limit-memory=MB] [--uncompress] [--backup-dir=PATH]
            [PROGRESS-REPORT-OPTIONS]
            apply-log

mysqlbackup [STD-OPTIONS]
            [--incremental-backup-dir=PATH] [--backup-dir=PATH]
            [--limit-memory=MB] [--uncompress]
            [PROGRESS-REPORT-OPTIONS]
            apply-incremental-backup
```

- `apply-log`

  Brings the InnoDB tables in the backup up-to-date, including any changes made to the data while the backup was running.

- `apply-incremental-backup`

  Brings the backup up-to-date using the data from an incremental backup.

Example 4.3 Apply Log to Full Backup

```
mysqlbackup --backup-dir=/path/to/backup apply-log
```

It reads the `backup-my.cnf` file inside *`backup-dir`* to understand the backup. The `my.cnf` default files have no effect other than supplying the `limit-memory=`*`MB`* value, which limits usage of memory while doing the `apply-log` operation.

Because the apply-log operation does not apply to incremental backups, no *`incremental-backup-dir`* is needed for this operation.

4.1.1.3 Restore an Existing Backup

Restores the data files from a backup to their original locations within the database server. The MySQL instance must be shut down first before a restore operation. The options `datadir`, `innodb_log_files_in_group`, and `innodb_log_file_size` must be specified either in the target server's configuration file, in the file specified by the `--defaults-file` option, or as command-line options. For usage and examples, see Chapter 5, *Recovering or Restoring a Database*.

```
mysqlbackup [STD-OPTIONS]
            [SERVER-REPOSITORY-OPTIONS]
            [--backup-dir=PATH]
            [PROGRESS-REPORT-OPTIONS]
            copy-back
```

- `copy-back`

  Restores files from a backup to their original locations within the MySQL server. The database must be shut down before this operation is performed.

4.1.1.4 Work with Single-File Backups

To simplify transfer and management of backup data, you can keep each backup in a single file (the backup image). The `backup-to-image` option performs a backup directly to a single file, or the options here can pack an existing backup into a single file or unpack a single-file backup to a full backup directory structure. For usage and examples, see Section 3.3.5, “Making a Single-File Backup”.

```
mysqlbackup [STD-OPTIONS]
            [--backup-image=IMAGE] [--backup-dir=PATH]
            [PROGRESS-REPORT-OPTIONS]
            image-to-backup-dir

mysqlbackup [STD-OPTIONS]
            [--backup-dir=PATH] [--backup-image=IMAGE]
            [PROGRESS-REPORT-OPTIONS]
            backup-dir-to-image

mysqlbackup [STD-OPTIONS]
            [--backup-image=IMAGE] [--src-entry=PATH]
            list-image

mysqlbackup [STD-OPTIONS]
            [--backup-image=IMAGE]
            [PROGRESS-REPORT-OPTIONS]
            validate

mysqlbackup [STD-OPTIONS]
            [--backup-image=IMAGE]
            [--backup-dir=PATH]
            [--src-entry=PATH] [--dst-entry=PATH]
            [PROGRESS-REPORT-OPTIONS]
            extract
```

- `image-to-backup-dir`

  Unpacks a single-file backup to a full backup directory structure. You specify the paths to both the image file and the destination directory in which to unpack. For usage and examples, see Section 3.3.5, “Making a Single-File Backup”.

- `backup-dir-to-image`

  Packs an existing backup into a single file. Specify a `--backup-image` value of `-` (standard output) to stream an existing backup directory structure to a tape device or a command that transfers the backup to another server. The `--backup-image` parameter is either `-` or an absolute path outside the `backup-dir` directory. For usage and examples, see Section 3.3.5, “Making a Single-File Backup”.

- `list-image`

  Display the contents of a single-file backup. Lists all files and directories in the image. The `--src-entry=name` can be used to list a specific file or directory. If the name is a directory, all its files and subdirectories inside the image are recursively listed. For usage and examples, see Section 3.3.5, “Making a Single-File Backup”.

- `validate`

  Verifies that a single-file backup is not corrupted, truncated, or otherwise damaged. This operation compares checksum values stored in the image file against the contents of the files. You might run it after transferring the image file to another system. To see the sequence of commands involved to produce and check a single-file backup, and the output for successful and unsuccessful checks, see Section C.4, “Validating a Single-File Backup Image”.

- `extract`

  Unpacks an individual file or directory from a single-file backup. For troubleshooting or restoration operations that do not require the full set of backup data. The resulting file or directory goes in the current directory, or in `backup-dir` if specified. All files and directory contents in the image with absolute path names are extracted into the same absolute path names on the local system. For usage and examples, see Section 3.3.5, “Making a Single-File Backup”.

  The `--src-entry=path` option can be used for selective extraction of a single file or single directory in image. Specify the path as it appears in the image.

  The `--dst-entry=path` option, along with `--src-entry=path` option can be used to extract a single file or single directory into a user-specified file or directory respectively. If the `--src-entry` option is used, but `--dst-entry` option is omitted, then the selected file or directory is extracted to the same path in the local file system.

  The default destination for the extract is the current working directory. It can be overridden by the `--backup-dir` option. All the files with relative pathnames in the image are extracted to pathnames relative to the destination directory.

  If the image contains some entries with absolute pathnames, those entries are extracted to the same absolute path names even if `--backup-dir` option is specified. The `--dst-entry` option must be used to relocate an absolute pathname.

4.1.2 Standard Options

The standard options are options of a general nature, or options that are not classified under any other specific option group.

- The following standard options also exist for the `mysql` command. Full descriptions for these options can be found in the MySQL reference manual, accessible through, e.g., Server Option and Variable Reference. These options must be specified ahead of any other `mysqlbackup` options, including the rest of the standard options:

```
--print-defaults            Print the program argument list and exit.
--no-defaults               Don't read default options from any option file.
--defaults-file=PATH        Only read default options from the given file. It has to be the
                            first option to be specified, if used.
```

```
--defaults-extra-file=PATH   Read this file after the global files are read.
--defaults-group-suffix=str  Also read option groups with the usual names and a suffix of str.
```

- The following options are also common between `mysqlbackup` and `mysql`, and full descriptions for them can be found in the MySQL reference manual, accessible through, e.g., Server Option and Variable Reference. However, `mysqlbackup` does not accept any short forms for these options as `mysql` does (for example, you must use `--help` instead of `-h` for `mysqlbackup`):

```
--help      Display help.
--version   Display version information
--verbose   Print more verbose information.
--debug     Print debug information.
```

- More standard options are available for `mysqlbackup`:

```
--force       Force operations such as overwrite files, create backup directory.
--trace=level Trace level of messages by mysqlbackup.
```

  `--force`: By default, some of the operations halt rather than overwrite any user data or log files when told to write to existing files. `--force` allows the overwriting of InnoDB data and log files during the `apply-log` and `apply-incremental-backup` operations and allows the replacing of an image file during an `backup-to-image` or `backup-dir-to-image` option. For all other operations, the `--force` option is rejected with an error message.

  `--trace=level`: Trace level of `mysqlbackup` messages. The permissible levels, in the order of increasing fineness, are:

  - 0 - INFO (information, warnings, errors)
  - 1 - FINE (verbose option is enabled)
  - 2 - FINER (debug option is enabled)
  - 3 - FINEST (includes all low level outputs)

  Default: 0

4.1.3 Connection Options

When `mysqlbackup` creates a backup, it sends SQL commands to MySQL server using a database connection. The general connection details are the same as described in Connecting to the MySQL Server in the MySQL Reference Manual.

As part of the `mysqlbackup` invocation, specify the appropriate `--user`, `--password`, `--port`, and/or `--socket` options that are necessary to connect to the MySQL server.

You can specify the following connection-specific options in the `[mysqlbackup]` or `[client]` sections of a MySQL configuration file, or through `mysqlbackup` command-line options. `mysqlbackup` reads your default configuration files and then the `my.cnf` file specified on the command line.

> **Note**
>
> - `mysqlbackup` reads only `--user`, `--password`, `--port`, and `--socket` options from the `[client]` group, and ignores any other connection options.
> - If you do not provide a value for the `--password`, the command prompts for one from the keyboard.
> - The `--host` option is allowed in the configuration file for compatibility, but currently it has no effect. The `mysqlbackup` command always connects to the local server's IP address.

```
Options Common to mysqld
========================

--port=port-num
--protocol=tcp|socket|pipe|memory
--pipe [ alias for --protocol=pipe ]
--user=name [ short option: -u ]
--host=hostname
--socket=name
--shared-memory-base-name=value [Windows only]
--character-sets-dir=PATH
--default-character-set=VALUE
--secure-auth [ Don't connect to pre-4.1.1 server ]
--password[=value]  [ short option: -p ]
--connect_timeout
--ssl [ Enalbe SSL for connection ]
--ssl-key=file_name
--ssl-cert=file_name
--ssl-ca=file_name
--ssl-capath=directory_name
--ssl-cipher=cipher_list
--ssl-verify-server-cert

Connection Options Specific to mysqlbackup
==========================================
--no-connection
--connect-if-online
```

Most other connection parameters used by the `mysql` command are recognized, but silently ignored. Unknown connection parameters cause the `mysqlbackup` command to stop.

The following connections options are specific to `mysqlbackup`:

- `--no-connection`

  The `--no-connection` option supersedes the other connection options and uses file-level operations to perform the backup. When you use this option, you must specify in the configuration file or on the command line many options whose values are normally retrieved automatically through the database connection.

  > **Warning**
  >
  > This option also turns on the `--no-history-logging` and `--no-locking` options, which might result in inconsistencies in non-InnoDB data if the tables are modified during the backup operation. It might also affect subsequent incremental backups; see the description for the `--incremental-base` option for details.

- `--connect-if-online`

  By default, a database connection is used for backup operations both during the initial stage to retrieve source repository configuration, and to lock tables while copying non-InnoDB data. This option allows `mysqlbackup` to make connection attempts in both phases, but continues even if the connection cannot be established. If a connection cannot be established, the processing is the same as with the `--no-connection` option. This option can be useful in emergency situations: for example, when the database server goes down during the backup operation.

4.1.4 Server Repository Options

The repository options specify various parameters related to the database server (the source) and the backup directory (the destination).

These options are used only with the following operations:

- Backup creation operations: `backup`, `backup-and-apply-log`, `backup-to-image`.

- Restore operations: `copy-back`.

When a database connection is available during a backup, the parameters describing the source repository are ignored, overridden by the corresponding values retrieved from the database connection.

The following parameters describe the Source Repository:

- `--datadir=`*`PATH`*
- `--innodb_data_file_path=`*`VALUE`* [Example: ibdata1:32M:autoextend]
- `--innodb_data_home_dir=`*`PATH`*
- `--innodb_log_group_home_dir=`*`PATH`*
- `--innodb_log_files_in_group=`*`N`*
- `--innodb_log_file_size=`*`SIZE`*

4.1.5 Backup Repository Options

These options specify various parameters related to the layout of the backup directory. Several of these option values can be derived automatically from the corresponding configuration option without the `backup` prefix, thus the `--backup-dir` option is the only one from this group that you typically specify.

These options are used only with the following operations:

- Backup creation operations: `backup`, `backup-and-apply-log`, `backup-to-image`.
- Restore operations: `copy-back`.

When a database connection is available during a backup, the parameters describing the source repository are ignored, overridden by the corresponding values retrieved from the database connection.

The following parameters describe the layout of files in the backup directory:

- `--backup-dir=`*`PATH`*

  The directory under which to store the backup data. This is a crucial parameter required for most kinds of backup operations. It cannot be a subdirectory of the directory specified by `--datadir`. An additional level of subdirectory is created when the `--with-timestamp` option is also specified.

- `--backup_innodb_data_file_path=`*`VALUE`* [Example: ibdata1:32M:autoextend]
- `--backup_innodb_data_home_dir=`*`PATH`*
- `--backup_innodb_log_group_home_dir=`*`PATH`*
- `--backup_innodb_log_files_in_group=`*`N`*
- `--backup_innodb_log_file_size=`*`SIZE`*
- `--backup_innodb_page_size=`*`SIZE`*
- `--backup_innodb_checksum_algorithm=`*`NAME`*
- `--with-timestamp`

  Creates a subdirectory underneath the backup directory, with a name formed from the timestamp of the backup operation. Useful to maintain a single backup directory containing many backup snapshots.

Default: no timestamped subdirectory is created. To reuse the same backup directory for a new backup, either remove the previous backup files manually or specify the `--force` [45] option to overwrite them.

4.1.6 Metadata Options

These options control the generation of metadata about backups. Some metadata is stored in the backup directory, other metadata is stored in tables within the `mysql` database of the backed-up instance.

- `--no-history-logging`

  Turns off the recording of backup progress and history in logging tables inside the backed-up database. See Section 9.4, “Using the MySQL Enterprise Backup Logs” for details about these tables.

  Default: history logging is enabled. When `--no-connection` is specified, history logging is automatically disabled. When `--connect-if-online` is specified, history logging only works if a database connection is successfully established during the backup.

- `--comments`=*`STRING`*

  Specify a comment string that describes or identifies the backup. Surround multi-word comments with appropriate quotation marks. The string is saved in a file `meta/comments.txt` in the backup. For example: `--comments="Backup of HR data on 2010/12/10"`.

- `--comments-file`=*`PATH`*

  Specify path to a file containing comments describing the backup. This file is saved as `meta/comments.txt` in the backup. For example: `--comments-file=/path/to/comments.txt`.

  This option overrides the `--comments` option if both are specified.

4.1.7 Compression Options

For instructions about using these options, see Section 3.3.3, “Making a Compressed Backup”.

- `--compress`

  Create backup in compressed format. For a regular backup, only the InnoDB data files are created in compressed format, using the `.ibz` extension. For a single-image backup, again only the InnoDB data files are compressed inside the backup image.

  Default: compression is disabled. Default compression level is 1 when compression is enabled. You can change the amount of compression with the `compress-level` option.

- `--compress-level`=*`LEVEL`*

  Specifies the level of compression. Value 0 disables compression. Value 1 is fastest compression, and value 9 is highest (and slowest) compression.

  Default: 1 (lowest and fastest compression). Explicitly specifying a non-zero value through configuration file or command line automatically enables the `--compress` option as well.

- `--uncompress`

  When used with the `apply-log` operation, uncompresses the compressed backup before applying the InnoDB log.

4.1.8 Incremental Backup Options

For an overview of incremental backups and usage information about these options, see Section 3.3.2, “Making an Incremental Backup”.

To take an incremental backup, specify the `--incremental` or `--incremental-with-redo-log-only`, along with the `--incremental-backup-dir`. All InnoDB data modified after the specified LSN is copied in the incremental backup. Depending on the choice of `--incremental` or `--incremental-with-redo-log-only` other options are required or recommended.

- `--incremental`

  Specifies that the associated `backup` or `backup-to-image` operation is incremental. Also requires either the `--incremental-base` option, or the combination of the `--start-lsn` and `--incremental-backup-dir` options.

  The incremental aspect applies only to InnoDB tables. By default, all non-InnoDB and `.frm` files are also included in incremental backup. To exclude non-InnoDB data in an incremental backup, use the `--only-innodb` or `--only-innodb-with-frm` option.

- `--incremental-with-redo-log-only`

  Specifies an alternative form of incremental backup for a `backup` or `backup-to-image` operation. Also requires either the `--incremental-base` option, or the combination of the `--start-lsn` and `--incremental-backup-dir` options.

  The incremental backup performed by this option has different performance characteristics and operational limitations than with the `--incremental` option:

  - The changes to InnoDB tables are determined based on the contents of the `InnoDB` redo log. Since the redo log files have a fixed size that you know in advance, it can require less I/O to read the changes from them than to scan the InnoDB tablespace files to locate the changed pages, depending on the size of your database, amount of DML activity, and size of the redo log files.

  - Since the redo log files act as a circular buffer, with records of older changes being overwritten as new DML operations take place, you must take new incremental backups on a predictable schedule that depends on the size of the log files and the amount of redo data generated for your workload. Otherwise, the redo log might not reach back far enough to record all the changes since the previous incremental backup. In this case, the `mysqlbackup` command quickly determines it cannot proceed and returns an error. Your backup script can catch the error and do an incremental backup with the `--incremental` option instead. options.

    For example:

    - To calculate the size of the redo log, issue the command `SHOW VARIABLES LIKE 'innodb_log_file%'`, and based on the output, multiply the `innodb_log_file_size` setting by `innodb_log_files_in_group`. To compute redo log size at the physical level, look in the `datadir` directory of the MySQL instance and sum the sizes of the files matching the pattern `ib_logfile*`.

    - The InnoDB LSN value corresponds to the number of bytes written to the redo log. To check the LSN at some point in time, issue the command `SHOW ENGINE INNODB STATUS` and look under the `LOG` heading. While planning your backup strategy, record the LSN values periodically and subtract the earlier value from the current one to calculate how much redo data is generated each hour, day, and so on.

    Prior to MySQL 5.5, it was common practice to keep the redo logs fairly small to avoid long startup times when the MySQL server was killed rather than shut down normally. In MySQL 5.5 and higher, the performance of crash recovery is significantly improved. With those releases, you can make your redo log files bigger if that helps your backup strategy and your database workload.

- This type of incremental backup is not so forgiving of too-low `--start-lsn` values as the standard `--incremental` option. For example, you cannot make a full backup and then make a series of `--incremental-with-redo-log-only` backups all using the same `--start-lsn` value. Make sure to specify the precise end LSN of the previous backup as the start LSN of the next incremental backup; do not use arbitrary values.

  > **Note**
  >
  > To ensure the LSN values match up exactly between successive incremental backups using this option, Oracle recommends always using the `--incremental-base` option when you use the `--incremental-with-redo-log-only` option.

- To judge whether this type of incremental backup is practical and efficient for a particular MySQL instance:

  - Measure how fast the data changes within the InnoDB redo log files. Check the LSN periodically to see how much redo data accumulates over the course of some number of hours or days.

  - Compare the rate of redo log accumulation with the size of the redo log files. Use this ratio to see how often to take an incremental backup, to avoid the likelihood of the backup failing due to historical data not available in the redo log. For example, if you are producing 1GB of redo log data per day, and the combined size of your redo log files is 7GB, you would schedule incremental backups more frequently than once a week. You might perform incremental backups every day or two, to avoid a potential issue if a sudden flurry of updates produced more redo than usual.

  - Benchmark incremental backup times using both the `--incremental` and `--incremental-with-redo-log-only` options, to confirm if the redo log backup technique performs faster and with less overhead than the traditional incremental backup method. The result could depend on the size of your data, amount of DML activity, and size of your redo log files; do your testing on a server with a realistic data volume and running a realistic workload. For example, if you have huge redo log files, reading them in the course of an incremental backup could take as long as reading the InnoDB data files using the traditional incremental technique. Conversely, if your data volume is large, reading all the data files to find the few changed pages could be less efficient than processing the much smaller redo log files.

  As with the `--incremental` option, the incremental aspect applies only to InnoDB tables. By default, all non-InnoDB and `.frm` files are also included in incremental backup. To exclude non-InnoDB data in an incremental backup, use the `--only-innodb` or `--only-innodb-with-frm` option.

- `--incremental-base=`*`mode`*`:`*`argument`*

  With this option, the `mysqlbackup` retrieves the information needed to perform incremental backups from the metadata inside the backup directory rather than from the `--start-lsn` option. It saves you from having to specify an ever-changing, unpredictable LSN value when doing a succession of incremental backups. Instead, you specify a way to locate the previous backup directory through the combination of *`mode`* and *`argument`* in the option syntax. The alternatives are:

  - `dir:`*`directory_path`*

    You specify the prefix `dir:` followed by a directory path. The path argument points to the root directory where the data from the previous backup is stored. With the first incremental backup, you specify the directory holding the full backup; with the second incremental backup, you specify the directory holding the first incremental backup, and so on.

  - `history:last_backup`

You specify the prefix `history:` followed by `last_backup`, the only valid argument for this mode. This makes `mysqlbackup` query the `end_lsn` value from the last successful backup as recorded in the `backup_history` table of the applicable instance.

> **Warning**
>
> Do not use the `history:` mode if the previous backup was a full backup taken with the `--no-connection` option, which always turns off the recording of backup history and might cause errors in a subsequent incremental backup using this mode of the `--incremental-base` option.

- `--start-lsn=`*`LSN`*

  In an incremental backup, specifies the highest LSN value included in a previous backup. You can get this value from the output of the previous backup operation, or from the `backup_history` table's `end_lsn` column for the previous backup operation. Always used in combination with the `--incremental` option; not needed when you use the `--incremental-base` option; not recommended when you use the `--incremental-with-redo-log-only` mechanism for incremental backups.

- `--incremental-backup-dir=`*`PATH`*

  Specifies the location under which to store data from an incremental backup. This is the same location you specify with `--incremental-base` if you use that option for a subsequent incremental backup.

Example 4.4 Incremental Backup

These examples show typical combinations of options used for incremental backups.

```
mysqlbackup --incremental \
  --incremental-backup-dir=/var/mysql/backup/latest \
  --incremental-base=dir:/var/mysql/backup/previous \
  ... backup

mysqlbackup --incremental-with-redo-log-only \
  --incremental-backup-dir=/var/mysql/backup/latest \
  --incremental-base=dir:/var/mysql/backup/previous \
  ... backup

mysqlbackup --incremental --start-lsn=12345 \
  --incremental-backup-dir=/var/mysql/backup/inc \
  ... backup

mysqlbackup --incremental-with-redo-log-only --start-lsn=12345 \
  --incremental-backup-dir=/var/mysql/backup/inc \
  ... backup
```

4.1.9 Partial Backup Options

For an overview of partial backups and usage information about these options, see Section 3.3.4, “Making a Partial Backup”.

- `--include=`*`REGEXP`*

  This option is for filtering InnoDB tables for backup. The InnoDB tables' fully qualified names are checked against the regular expression specified by the option. If the REGEXP matches *`db_name.table_name`*, the table is included. The regular expression syntax is the extended form specified in the POSIX 1003.2 standard. For example, `--include=mydb\.t[12]` matches the tables `t1` and `t2` in the database `mydb`.

  This option only applies to InnoDB tables created with the MySQL option `innodb_file_per_table` enabled (which is the default setting for MySQL 5.6 and after), in which

case the tables are in separate files that can be included or excluded from the backup. All tables in the InnoDB system tablespace are always backed up.

Default: Backs up all InnoDB tables.

> **Note**
>
> This option does not filter non-InnoDB tables, for which options like `--databases` and `--databases-list-file` can be used.

> **Important**
>
> This option does not filter the `.frm` files associated with InnoDB tables, meaning that regardless of the option's value, all the `.frm` files for all InnoDB tables are always backed up unless they are excluded by other options. Those `.frm` files for InnoDB tables that are not backed up should be deleted before the database backup is restored. See Section 3.3.4.1, "Backing Up Some or All InnoDB Tables" for details.

- `--databases=`*`LIST`*

  Specifies the list of non-InnoDB tables to back up. The argument specifies a space-separated list of database or table names of the following form:

```
"db_name[.table_name] db_name1[.table_name1] ...".
```

  If the specified values do not match any database or table, then no non-InnoDB data files are backed up. See Section 3.3.4.3, "Backing Up Non-InnoDB Tables from Selected Databases" for details.

  By default, all non-InnoDB tables from all databases are backed up.

> **Note**
>
> The option has no filtering effects on the InnoDB data files (`.ibd` files) for the databases or tables it specifies. To filter InnoDB data files, use the `--include` option instead.

- `--databases-list-file=`*`PATH`*

  Specifies the pathname of a file that lists the non-InnoDB tables to be backed up. The file contains entries for databases or fully qualified table names separated by newline or space. The format of the entries is the same as for the `--databases` option:

```
db_name[.table_name]
db_name1[.table_name1]
...
```

  Remove any whitespaces surrounding the database or table names, as the whitespaces are not removed automatically. Begin a line with the `#` character to include a comment. No regular expressions are allowed.

  If the specified entries do not match any database or table, then no non-InnoDB data files are backed up.

> **Note**
>
> The option has no filtering effects on the InnoDB data files (`.ibd` files) for the databases or tables it specifies. To filter InnoDB data files, use the `--include` option instead.

- `--only-known-file-types`

By default, all files in the data directory are included in the backup. (See Section 1.4, “Files that Are Backed Up” for details.) If the `--only-known-file-types` option is specified, the backup includes only the files with these file extensions:

- `.ARM`: Archive storage engine metadata.
- `.ARZ`: Archive storage engine data.
- `.CSM`: CSV storage engine data.
- `.CSV`: CSV storage engine data.
- `.frm`: table definitions.
- `.MRG`: Merge storage engine references to other tables.
- `.MYD`: MyISAM data.
- `.MYI`: MyISAM indexes.
- `.OPT`: database configuration. information
- `.PAR`: partition definitions.
- `.TRG`: trigger parameters.
- `.TRN`: trigger namespace information.

- `--only-innodb`

  Back up only InnoDB data and log files. All `.frm` files and files created by other storage engines are excluded. Typically used when no connection to `mysqld` is allowed or when there is no need to copy MyISAM or `.frm` files, for example, when you are sure there are no DDL changes during the backup. See Section 3.3.4.1, “Backing Up Some or All InnoDB Tables” for instructions and examples.

  Can be used in combination with the `--suspend-at-end` option to allow customized scripting at the end of backup. Not compatible with the `--slave-info` option.

  Default: backups include files from all storage engines.

- `--only-innodb-with-frm[={all|related}]`

  Back up only InnoDB data, log files, and the `.frm` files associated with the InnoDB tables.

  - `--only-innodb-with-frm=all` includes the `.frm` files for all InnoDB tables in the backup.
  - `--only-innodb-with-frm=related`, in combination with the `--include` option, copies only the `.frm` files for the tables that are included in the partial backup.
  - `--only-innodb-with-frm` with no argument is the same as `--only-innodb-with-frm=related`.

  > **Note**
  >
  > For incremental backups, even only changed `.ibd` files are backed up, `.frm` files associated with *all* specified InnoDB tables are included.

  This option saves you having to script the backup step for InnoDB `.frm` files, which you would normally do while the server is put into a read-only state by a `FLUSH TABLES WITH READ LOCK` statement. The `.frm` files are copied without putting the server into a read-only state, so that the

backup operation is a true hot backup and does not interrupt database processing. You must ensure that no `ALTER TABLE` or other DDL statements change `.frm` files for InnoDB tables while the backup is in progress. If the `mysqlbackup` command detects changes to any relevant `.frm` files during the backup operation, it halts with an error. If it is not practical to forbid DDL on InnoDB tables during the backup operation, use the `--only-innodb` option instead and use the traditional method of copying the `.frm` files while the server is locked.

All files created by other storage engines are excluded. Typically used when no connection to `mysqld` is allowed or when there is no need to copy MyISAM files, for example, when you are sure there are no DDL changes during the backup. See Section 3.3.4.1, “Backing Up Some or All InnoDB Tables” for instructions and examples.

Can be used in combination with the `--suspend-at-end` option to allow customized scripting at the end of backup. Not compatible with the `--slave-info` option.

Default: backups include files from all storage engines.

4.1.10 Single-File Backup Options

These options are associated with single-file backups. You use them in combination with the `mysqlbackup` subcommands `backup-to-image`, `image-to-backup-dir`, `backup-dir-to-image`, `list-image`, and `extract` that pack or unpack single-image backups. For usage information, see Section 3.3.5, “Making a Single-File Backup”.

- `--backup-image=`*`IMAGE`*

  Specify the path name of the file used for a single-file backup. By default, the single-file backup is streamed to standard output, so that you can pipe it directly to other commands such as tape backup or `ssh`-related network commands.

  You can optionally prefix the image name with `file:` to signify file I/O (the default). For tape backups, prefix the image name with`sbt:`. See Section 3.3.5.2, “Backing Up to Tape” for details about tape backups.

- `--src-entry=`*`PATH`*

  Identifies a file or directory to extract from a single-file backup. This option is used with the `extract` command. If the argument is a directory, all its files and subdirectory contents are extracted. No pattern matching expression is allowed for the argument. Optionally, you can also specify the `--dst-entry` option to extract the file or directory in a location different from its original path name.

  For example: `src-entry=meta/comments.txt` extracts only one file, `comments.txt`, while `src-entry=meta` extracts the entire directory tree for the `meta` subdirectory.

  Default: All entries are extracted.

- `--dst-entry=`*`PATH`*

  Used with single-file backups to extract a single file or directory to a user-specified path. Use of this option requires specifying the `--src-entry` option. This option specifies the destination path for the selected entry in backup image corresponding to entry specified by `-src-entry=`*`PATH`* option. The entry could point to a single file or single directory. For example, to retrieve the comments file from a backup image and store it as `/tmp/my-comments.txt`, use a command like the following:

```
mysqlbackup --src-entry=meta/comments.txt \
  --dst-entry=/tmp/my-comments.txt \
  --backup-image=/var/myimage.bki  extract
```

  Similarly, to extract all the contents of the `meta` directory in a single-file backup as `/data/my-meta`, use a command like the following:

```
mysqlbackup --src-entry=meta \
  --dst-entry=/data/my-meta \
  --backup-image=/var/myimage.bki  extract
```

The specified path is a simple path name without any wildcard expansion or regular expressions.

Default: By default, original pathnames are used to create files in the local file system.

- `--sbt-database-name=`*`NAME`*

  For tape backups, this option can be used as a hint to the Media Management Software (MMS) for the selection of media and policies. This name has nothing to do with MySQL database names. It is a term used by the MMS. See Section 3.3.5.2, “Backing Up to Tape” for usage details.

- `--sbt-lib-path=`*`PATH`*

  Path name of the SBT library used by software that manages tape backups. If this is not specified, operating system-specific search methods are used to locate `libobk.so` (UNIX) or `orasbt.dll` (Windows). See Section 3.3.5.2, “Backing Up to Tape” for usage details.

- `--sbt-environment=`*`VAR`*`=`*`value`*`,...`

  Passes product-specific environment variables to Oracle Secure Backup or another SBT-compliant backup management product, as an alternative to setting and unsetting environment variables before and after each `mysqlbackup` invocation.

  The parameter to this option is a comma-separated list of key-value pairs, using syntax similar to that of the RMAN tool for the Oracle Database. For example, `--sbt-environment=VAR1=val1,VAR2=val2,VAR3=val3`.

  Consult the documentation for your backup management product to see which of its features can be controlled through environment variables. For example, the Oracle Secure Backup product defines environment variables such as `OB_MEDIA_FAMILY`, `OB_DEVICE`, and `OB_RESOURCE_WAIT_TIME`. You might set such variables with the `mysqlbackup` by specifying an option such as `--sbt-environment="OB_MEDIA_FAMILY=my_mf,OB_DEVICE=my_tape"`.

  If the argument string contains any whitespace or special characters recognized by the command shell, enclose the entire argument string in quotation marks. To escape an equals sign or comma, use the `\` character. For example, `--sbt-environment="VAR1=multiple words,VAR2=<angle_brackets>,VAR3=2+2\=4"`.

- `--disable-manifest`

  Disable generation of manifest files for a backup operation, which are `backup_create.xml` and `backup_content.xml` present in the `meta` subdirectory.

4.1.11 Performance / Scalability / Capacity Options

These options limit the resources used by the backup process, in order to minimize backup overhead for busy or huge databases, or specify behaviors of the process when encountering resource issues.

- `--number-of-buffers=`*`num_buffers`*

  Specifies the number of buffers, each 16MB in size, to use during multithreaded options.

  Use a high number for CPU-intensive processing such as backup, particularly when using compression. Use a low number for disk-intensive processing such as restoring a backup. This value should be at least as high as the number of read threads or write threads, depending on the type of operation.

Default: computed internally depending on the available memory and the type of operation. The basic formula is:

```
(read threads + write threads + process threads + max(read threads, write threads, process threads))
```

For compression or incremental backup operations, the buffer size is slightly more than 16MB to accommodate the headers.

One additional buffer is used for single-file incremental backup and single-file compressed backup.

Compressed backup, compressed single-file backup, and uncompress apply-log operations require one additional buffer for each process thread.

If you change the number of read, write, and processing threads, you can experiment with changing this value so that it is slightly larger than the total number of threads specified by those other options. See Section 7.1, “Optimizing Backup Performance” and Section 7.2, “Optimizing Restore Performance” for additional advice about recommended combinations of values for this and other performance-related options for various hardware configurations, such as RAID or non-RAID storage devices.

- `--read-threads=num_threads`

  Specifies the number of threads to use for reading data from disk.

  Default: currently 3. This default applies to these kinds of operations: `copy-back`, `extract`, and `backup`. If you specify a value of 0, it is silently adjusted to 1. The maximum is 15; if you supply a negative value, it is silently adjusted to 15. For `apply-log` operations, the number of read threads is always 1 regardless of this option setting. See Section 7.1, “Optimizing Backup Performance” and Section 7.2, “Optimizing Restore Performance” for advice about recommended combinations of values for `--read-threads`, `--process-threads`, and `--write-threads` for various hardware configurations, such as RAID or non-RAID storage devices.

- `--process-threads=num_threads`

  Specifies the number of threads to use for processing data, such as compressing or uncompressing backup files.

  Default: currently 3. This default applies to these kinds of operations: `extract`, and `backup`. It is ignored when you use any of the options `--incremental-with-redo-log-only`, `apply-incremental-backup`, `copy-back`, or `backup-dir-to-image`.

  If you specify a value of 0, it is silently adjusted to 1. The maximum is 15; if you supply a negative value, it is silently adjusted to 15. For `apply-log` operations, the number of process threads is always 1 regardless of this option setting. See Section 7.1, “Optimizing Backup Performance” and Section 7.2, “Optimizing Restore Performance” for advice about recommended combinations of values for `--read-threads`, `--process-threads`, and `--write-threads` for various hardware configurations, such as RAID or non-RAID storage devices.

- `--write-threads=num_threads`

  Specifies the number of threads to use for writing data to disk.

Default: currently 3. This default applies to these kinds of operations: `copy-back`, `extract`, and `backup`. It is ignored when you use any of the single-file backup options `list-image` or `validate`.

If you specify a value of 0, it is silently adjusted to 1. The maximum is 15; if you supply a negative value, it is silently adjusted to 15. For `apply-log` operations, the number of write threads is always 0 regardless of this option setting. See Section 7.1, “Optimizing Backup Performance” and Section 7.2, “Optimizing Restore Performance” for advice about recommended combinations of values for `--read-threads`, `--process-threads`, and `--write-threads` for various hardware configurations, such as RAID or non-RAID storage devices.

- `--limit-memory=`*`MB`*

  Specify maximum memory in megabytes that can be used by the `mysqlbackup` command. Formerly applied only to `apply-log` operation, but in MySQL Enterprise Backup 3.8 and higher it applies to all operations. Do not include any suffixes such as `mb` or `kb` in the option value.

  Default: 100 for `apply-log` operations; 300 for all other operations. megabytes).

  The memory limit specified by this option also caps the number of 16MB buffers available for multithreaded processing. For example, with a 300 MB limit, the maximum number of buffers is 18. If additional buffers are required because you increase the values for `--read-threads`, `--process-threads`, `--write-threads`, and/or `--number-of-buffers`, increase the `--limit-memory` value proportionally.

- `--sleep=`*`MS`*

  Specify the number in milliseconds to sleep after copying a certain amount of data from InnoDB tables. Each block of data is 1024 InnoDB data pages, typically totalling 16MB. This is to limit the CPU and I/O overhead on the database server.

  Default: 0 (no voluntary sleeps).

- `--no-locking`

  Disables locking during backup of non-InnoDB files, even if a connection is available. Can be used to copy non-InnoDB data with less disruption to normal database processing. There could be inconsistencies in non-InnoDB data if any changes are made while those files are being backed up.

- `--page-reread-time=`*`MS`*

  Interval in milliseconds that `mysqlbackup` waits before re-reading a page that fails a checksum test. A busy server could be writing a page at the same moment that `mysqlbackup` is reading it. Can be a floating-point number, such as 0.05 meaning 50 microseconds. Best possible resolution is 1 microsecond, but it could be worse on some platforms. Default is 100 milliseconds (0.1 seconds).

- `--page-reread-count=`*`retry_limit`*

  Maximum number of re-read attempts, when a page fails a checksum test. A busy server could be writing a page at the same moment that `mysqlbackup` is reading it. If the same page fails this many checksum tests consecutively, with a pause based on the `--page-reread-time` option between each attempt, the backup fails. Default is 500.

- `--on-disk-full={abort|abort_and_remove|warn}`

  Specifies the behavior when a backup process encounters a disk-full condition. This option is only for backup operations (`backup`, `backup-and-apply-log`, and `backup-to-image`).

  - `abort`: Abort backup, without removing the backup directory. The disk remains full.
  - `abort_and_remove`: Abort backup and remove the backup directory.

- `warn`: Write a warning message every 30 seconds and retry backup until disk space becomes available.

Default: `abort`.

4.1.12 Progress Report Options

There are two options for controlling the progress reporting function of `mysqlbackup`: `--show-progress` and `--progress-interval`:

- `--show-progress[={stderr|stdout|file:FILENAME|fifo:FIFONAME|table|variable}]`

  The option instructs `mysqlbackup` to periodically output short progress reports known as progress indicators on its operation.

  The argument of the option controls the destination to which the progress indicators are sent:

  - `stderr`: Progress indicators are sent to the standard error stream. The report is embedded in a time-stamped `mysqlbackup` INFO message. For example:

    ```
    130607 12:22:38 mysqlbackup: INFO: Progress: 191 of 191 MB; state: Completed
    ```

  - `stdout`: Progress indicators are sent to the standard output stream. A single newline character is printed after each progress indicator.

  - `file:FILENAME`: Progress indicators are sent to a file. Each new progress report overwrites the file, and the file contains the most recent progress indicator followed by a single newline character.

  - `fifo:FIFONAME`: Progress indicators are sent to a file system FIFO. A single newline character is printed after each progress indicator.

    > **Warning**
    >
    > If there is no process reading the FIFO, the `mysqlbackup` process hangs at the end of the execution.

  - `table`: Progress indicators are sent to the `mysql.backup_progress` table. This requires a connection to the MySQL server, and therefore, only works when backing up a running MySQL instance. `mysqlbackup` first adds one row of the progress report to the `mysql.backup_progress` table, and then updates the row afterwards with the latest progress indicator. The progress indicator is stored in the `current_status` column of the table.

    If the backup locks the MySQL instance (for example, by issuing a `FLUSH TABLES WITH READ LOCK` statement), the progress reports are not delivered to the `mysql.backup_progress` table until the MySQL instance is unlocked.

  - `variable`: Progress indicators are sent to the system variable `backup_progress`.

    > **Warning**
    >
    > The system variable `backup_progress` is not yet defined for the MySQL Server. Users need to create their own plugin to define the variable. See The MySQL Plugin API for more information on user plugins.

  When there is no argument specified for `--show-progress`, progress indicators are sent to `stderr`.

  Progress can be reported to multiple destinations by specifying the `--show-progress` option several times on the command line. For example the following command line reports progress of the backup command to `stderr` and to a file called `meb_output`:

```
mysqlbackup --show-progress --show-progress=file:meb_output --backup-dir=/full-backup
            backup
```

The progress indicators are short strings that indicate how far the execution of a `mysqlbackup` operation has progressed. A progress indicator consists of one or more meters that measure the progress of the operation. For example:

```
Progress: 100 of 1450 MB; state: Copying .ibd files
```

This shows that 100 megabytes of a total of 1450 megabytes have been copied or processed so far, and `mysqlbackup` is currently copying InnoDB data files (`.ibd` files).

The progress indicator string begins with `Progress:`, followed by one or more meters measuring the progress. If multiple meters are present, they are separated by semicolons. The different types of meters include:

- Total data meter: It is always the first meter in the progress indicator. It is in the format of:

  ```
  DATA of TOTAL UNIT
  ```

  DATA and *TOTAL* are unsigned decimal integers, and *UNIT* is either MB (megabytes), KB (kilobytes), or bytes (1MB=1024KB and 1KB=1024 bytes).

  The total data meter has two slightly different meanings depending on the `mysqlbackup` operation:

  - The amount of data copied or processed and the total amount of data to be copied or processed by the `mysqlbackup` operation. For example:

    ```
    Progress: 200 of 1450 MB
    ```

    When the operation is for, e.g., `backup`, the indicator means 200MB is copied of 1450MB. But when the operation is for, e.g., `validate` or `incremental`, it means 200MB is processed out of 1450MB.

  - Total amount of data copied or processed and an estimate for the total that will be copied by the end of the operation. The estimated total is updated as per the data on the server, as the execution of the command progresses.

    For some operations such as `backup`, it is not possible to know exactly at the start of the execution how much data will be copied or processed. Therefore, the total data meter shows the estimated amount of the total data for a backup. The estimate is updated during the execution of the command. For example:

    ```
    Progress: 200 of 1450 MB
    ```

    is followed by:

    ```
    Progress: 200 of 1550 MB
    ```

    when 100MB of data is added on the server.

    If the operation is successful, the final progress indicator shows the actual amount of data copied at the end of the operation.

- Compression meter: It indicates the sliding average of the compression ratio, which is defined for each block of data that is compressed as `(orig_size - compressed_size) / orig_size`. For example:

  ```
  compression: 40%
  ```

  This means that after compression, the data takes 40% less space (calculated as an average over the last 10 data blocks).

The compression meter is included in the progress indicator if the `--compress` option is enabled for the `mysqlbackup` operation. The value of the compression meter is undefined until at least 10 data blocks have been compressed. The undefined meter value is denoted by the '-' in the meter:

```
compression: -
```

- State meter: It is a short description of the major step the command is currently executing. For example:

```
state: Copying InnoDB data
```

```
state: Waiting for locks
```

```
state: Copying system tablespace
```

```
state: Copying .ibd files
```

```
state: Copying non-InnoDB data
```

```
state: Completed
```

Here are some examples of progress indicators with different meters:

```
Progress: 300 of 1540 MB; state: Waiting for locks
```

```
Progress: 400 of 1450 MB; state: Copying InnoDB data: compression: 30%
```

The exact set of meters included in the progress indicator depends on the command and the options used for it.

- `--progress-interval=`*`SECONDS`*

  Interval between progress reports in seconds. Default value is two seconds. The shortest interval is 1 second and the longest allowed interval is 100000 seconds.

4.1.13 Options for Special Backup Types

These options are for backing up database servers that play specific roles in replication, or contain certain kinds of data that require special care in backing up.

- `--slave-info`

  When backing up a replication slave server, this option captures information needed to set up an identical slave server. It creates a file `meta/ibbackup_slave_info` inside the backup directory, containing a `CHANGE MASTER` statement with the binary log position and name of the binary log file of the master server. This information is also printed in the `mysqlbackup` output. To set up a new slave for this master, restore the backup data on another server, start a slave server on the backup data, and issue a `CHANGE MASTER` command with the binary log position saved in the `ibbackup_slave_info` file. See Section 6.1, “Setting Up a New Replication Slave” for instructions.

  > **Note**
  >
  > Only use this option when backing up a slave server. Its behavior is undefined when used on a master or non-replication server.
  >
  > This option is not compatible with the `--only-innodb` or `--only-innodb-with-frm` options.

- `--suspend-at-end`

  This option pauses the `mysqlbackup` command when the backup procedure is close to ending. It creates a file called `ibbackup_suspended` in the backup log group home directory and waits until

you delete that file before proceeding. This option is useful to customize locking behavior and backup of non-InnoDB files through custom scripting.

All tables are locked before suspending, putting the database into a read-only state, unless you turn off locking with the `--no-locking` or `--no-connection` option. The `--only-innodb` and `--only-innodb-with-frm` options also prevent the locking step. Because locking all tables could be problematic on a busy server, you might use a combination of `--only-innodb` or `--only-innodb-with-frm` and `--suspend-at-end` to back up only certain non-InnoDB tables.

- `--exec-when-locked`="*`utility arg1 arg2`* ..."

  You can use this option to write a script that backs up any information that is not part of the usual backup, for example by using `mysqldump` to back up tables from the MEMORY storage engine that are not on disk.

  Within your script, the `BACKUP_DIR` environment variable is set and points to the current backup directory. Use single quotes to prevent premature expansion of `$BACKUP_DIR`, as in the following examples.

  On Unix or Linux systems:

  ```
  mysqlbackup --exec-when-locked='mysqldump mydb t1 > $BACKUP_DIR/t1.sql' other_options
  ```

  Or on Windows systems:

  ```
  mysqlbackup --exec-when-locked="mysqldump mydb t1 > %BACKUP_DIR%/t1.sql" other_options
  ```

  If the utility cannot be executed or returns a non-zero exit status, then the whole backup process is cancelled. If you also use the `--suspend-at-end` option, the utility specified by `--exec-when-locked` is executed after suspending.

4.2 Configuration Files and Parameters

You can specify many `mysqlbackup` options either on the command line or as configuration parameters inside a configuration file. This section describes the use of configuration files and the meanings of the configuration options. For options that are typically specified on the command line, the primary descriptions and examples are in Section 4.1, “`mysqlbackup` Command-Line Options”.

In general, `mysqlbackup` follows the `mysql` style of processing configuration options: `[mysqlbackup]` and `[client]` group options are passed as command-line options. Any command-line options that you specify override the values from the configuration file, and in the case of duplicate options, the last instance takes precedence. `mysqlbackup` also reads options in the `[mysqld]` group to detect parameters related to the source repository when no connection to `mysqld` is available.

Options Files

The `mysqlbackup` command reads the location of the MySQL data to back up from (in order of priority):

- The connection information from the running database, whenever possible. Thus, in most cases, you can avoid specifying most options on the command line or in a configuration file.
- Parameters you specify on the `mysqlbackup` command line. You can specify certain options for individual backup jobs this way.
- The MySQL configuration file (by default, `my.cnf` on Unix and `my.ini` on Windows). The parameters are searched for first under the `[mysqlbackup]` group, then under the `[client]` group. You can put common parameters that apply to most of your backup jobs into the configuration file.

Because `mysqlbackup` **does not overwrite any files** during the initial backup step, the backup directory must not contain any old backup files. `mysqlbackup` stops when asked to create a file that already exists, to avoid harming an existing backup. For convenience, specify the `--with-timestamp` option, which always creates a unique timestamped subdirectory for each backup job underneath the main backup directory.

Configuration Files Stored with the Backup Data

Each set of backup data includes a configuration file, `backup-my.cnf`, containing a minimal set of configuration parameters. The `mysqlbackup` command generates this file to record the settings that apply to this backup data. Subsequent operations, such as the `apply-log` process, read options from this file to determine how the backup data is structured.

Example 4.5 Example `backup-my.cnf` file

Here is an example `backup-my.cnf` file generated by `mysqlbackup`:

```
[mysqld]
innodb_data_file_path=ibdata1:256M;ibdata2:256M:autoextend
innodb_log_file_size=256M
innodb_log_files_in_group=3
```

All paths in the generated `backup-my.cnf` file point to a single backup directory. For ease of verification and maintenance, you typically store all data for a backup inside a single directory rather than scattered among different directories.

During a backup, the configuration parameters that are required for later stages (such as the restore operation) are recorded in the `backup-my.cnf` file that is generated in the backup directory. Only the minimal required parameters are stored in `backup-my.cnf`, to allow you to restore the backup to a different location without extensive changes to that file. For example, although the `innodb_data_home_dir` and `innodb_log_group_home_dir` options can go into `backup-my.cnf`, they are omitted when those values are the same as the `backup-dir` value.

4.2.1 Source Repository Parameters

The following parameters are supported in configuration files under the `[mysqlbackup]` group. The underscore characters in parameter names can be replaced with dashes and treated as synonyms, similar to `mysqld` parameters that use this same convention. (See Using Options on the Command Line in the MySQL Reference Manual for details.) The documentation typically lists the names with underscores, to match the output of the `SHOW VARIABLES` statement.

For information about how these options are specified for the MySQL server, click the option name to see the description in the MySQL Reference Manual.

- `datadir`

  This is the `datadir` value used by the MySQL instance. The `.frm` files live here inside subdirectories named after the databases inside the instance.

  When a database connection exists, the value is retrieved automatically and overrides any value you specify. This is a crucial parameter for both the MySQL server and MySQL Enterprise Backup.

- `innodb_data_home_dir`

  Specifies the directory where InnoDB data files live. Usually the same as `datadir`, but can be different.

  This parameter, together with `innodb_data_file_path`, determines where the InnoDB data files such as `ibdata1`, `ibdata2`, and so on, are situated within the MySQL server.

  Typically, you do not need to specify this option, because its value is retrieved automatically using the database connection.

Its value is derived as follows:

- If `innodb_data_home_dir` is not specified, it inherits the value of `datadir`.
- If `innodb_data_home_dir` is a relative path, that path is located relative to (that is, underneath) the `datadir` value.
- An `innodb_data_home_dir` of `""` refers to the `/` root directory.
- If `innodb_data_home_dir` is an absolute path, its value is used as-is.

- `innodb_data_file_path`

  Specifies InnoDB data file names and sizes. Examples:

  ```
  ibdata1:32M;ibdata2:32M:autoextend
  /abs/path/ibdata1:32M:autoextend
  innodb-dir/ibdata1:32M:autoextend
  ```

  When a database connection exists, the value is retrieved automatically and overrides any value you specify.

  This parameter together with innodb_data_home_dir determines where the InnoDB data files (such as `ibdata1`, `ibdata2`, and so on) live in server repository.

  Typically, you do not need to specify this option, because its value is retrieved automatically using the database connection. If no database connection is available, you must specify it.

  Whether the initial filename begins with a `/` character or not, the files are located relative to the `innodb_data_home_dir` value.

- `innodb_log_group_home_dir`

  Specifies where InnoDB logs live within the server repository. Usually the same as `datadir`, but can be different.

  Its value is derived as follows:

  - If `innodb_log_group_home_dir` is not specified, it inherits the value of `datadir`.
  - If `innodb_log_group_home_dir` is a relative path, that path is located relative to (that is, underneath) the `datadir` value.
  - If `innodb_log_group_home_dir` is an absolute path, its value is used as-is.

- `innodb_log_files_in_group`

  Specifies the number of InnoDB log files before being rotated.

  Typically, you do not need to specify this option, because its value is retrieved automatically using the database connection. If no database connection is available, you must specify it.

  When a database connection exists, the value is retrieved automatically and overrides any value you specify.

- `innodb_log_file_size`

  Specifies maximum single InnoDB log file size before switching to next log file. Example: 20M.

  Typically, you do not need to specify this option, because its value is retrieved automatically using the database connection. If no database connection is available, you must specify it.

When a database connection exists, the value is retrieved automatically and overrides any value you specify.

4.2.2 Backup Repository Parameters

The following parameters are supported in configuration files under the `[mysqlbackup]` group. The underscore characters in parameter names can be replaced with dashes and treated as synonyms, similar to `mysqld` parameters that use this same convention. (See Using Options on the Command Line in the MySQL Reference Manual for details.) The documentation typically lists the names with underscores, to match the output of the `SHOW VARIABLES` statement.

The parameters marked as having “No Default” value are specified through `my.cnf` files, command-line parameters, or can be obtained automatically once the `mysqlbackup` command establishes a database connection.

- `backup_dir`

  The location under which backup destination files go. Typically retrieved automatically through the database connection. Must be specified if a database connection is not available. Same as the `--backup-dir` command-line option.

- `backup_innodb_data_home_dir`

  Specifies the directory where backup InnoDB data files live. Usually same as `backup-dir`, but can be different.

  This parameter together with `backup_innodb_data_file_path` determines where the InnoDB data files (such as `ibdata1`, `ibdata2`, ...) are stored inside the backup directory structure.

  This parameter is applicable only for backup operations, not restore.

  For the backup operations (such as `backup`, `backup-and-apply-log`, `backup-to-image`), the value of the backup destination directory is derived as follows:

  - If `backup_innodb_data_home_dir` is not specified, it inherits the value of `backup-dir`.
  - If `backup_innodb_data_home_dir` is a relative path, that path is located relative to (that is, underneath) the `backup-dir` value.
  - An `backup_innodb_data_home_dir` of `""` refers to the `/` root directory.
  - If `backup_innodb_data_home_dir` is an absolute path, its value is used as-is.

  To make it easy to relocate the backup directory and avoid editing the `backup-my.cnf` file, the backup operation writes this value into `backup-my.cnf` only if it is different than the `backup-dir` value, and using a relative path if possible.

  For `backup-to-image` operations, the final value of the `backup_innodb_data_home_dir` option must be a relative path, so that the single-file backup is machine-independent.

- `backup_innodb_data_file_path`

  Specifies InnoDB data file names and sizes. Examples:

  ```
  ibdata1:32M;ibdata2:32M:autoextend
  /abs/path/ibdata1:32M:autoextend
  innodb-dir/ibdata1:32M:autoextend
  ```

  This parameter together with `backup_innodb_data_home_dir` determines where the InnoDB data files (such as `ibdata1`, `ibdata2`, ...) live in the backup repository.

Within the backup directory, any data files specified with relative paths are located relative to the `backup_dir` path. Any data files specified with absolute paths are placed inside the `backup_innodb_data_home` directory.

When the parameter is not specified, it inherits the value from the value of the `innodb_data_file_path` option. If both source and destination attempt to use an absolute path that resolve to the same files, the backup is cancelled.

To specify absolute paths for InnoDB datafiles in backup, you must also set the `backup_innodb_data_home` option to `""`.

- `backup_innodb_log_group_home_dir`

  Specifies where backup InnoDB logs live. Usually the same as `backup-dir`, but can be different.

  The names of the log files are fixed and not reconfigurable.

  This parameter is applicable only for backup operations (not restore).

  The backup operation uses this value and writes it as `innodb_log_group_home_dir=value` in `backup-my.cnf`.

  For `copy-back` and `apply-log` operations, `innodb_log_group_home_dir` in `backup-my.cnf` is treated in a way that is compatible with how it was created.

- `backup_innodb_log_files_in_group`

  Specifies the number of InnoDB log files in backup before being rotated. Example: 5.

  Usually same as `innodb_log_files_in_group`, but can be different.

  The value for this parameter is derived as:

  - Specified `backup_innodb_log_files_in_group` value from command line or configuration file.
  - Else `innodb_log_files_in_group` value from the database connection, if available.
  - Else the `innodb_log_files_in_group` value from the command line or configuration file.

- `backup_innodb_log_file_size`

  Specifies maximum single InnoDB log file size in backup before switching to next log file. Example: 20M.

  Usually the same as `innodb_log_file_size`, but can be different.

  The value for this parameter is derived as:

  - Specified `backup_innodb_log_file_size` value from command line or configuration file.
  - Else `innodb_log_file_size` value from database connection, if available.
  - Else specified `innodb_log_file_size` value from command line or configuration file.

- `incremental-backup-dir`

  Specifies backup destination directory for incremental backup. Default: No Default.

- `backup-image`

Specifies the path for a single-file backup. Specifying any non-seekable device is also OK. The value `-` specifies standard output (`stdout`).

If the path is relative, it is interpreted relative to the `backup-dir` value. The extension `.mbi` extension that we use in documentation examples is not required.

4.2.3 Other Parameters

- `compress`

  Generates a compressed backup. Same as the `--compress` option.

- `compress-level`

  Specifies the level of compression, 0 (none) to 9 (maximum). Same as the `--compress-level` option.

- `only-innodb`

  Back up only InnoDB data and log files. Same as the `--only-innodb` option.

- `only-innodb-with-frm`

  Back up only InnoDB data, log files, and `.frm` files associated with InnoDB tables. Same as the `--only-innodb-with-frm` option.

- `no-history-logging`

  Turns off the recording of backup progress and history in logging tables inside the backed-up database. Same as the `--no-history-logging` option.

- `no-locking`

  Disables locking during backup of non-InnoDB files, even if a connection is available. Same as the `--no-locking` option.

- `no-connection`

  Prohibits making a connection to the `mysqld` server, for compatibility with previous behavior of the `ibbackup` command. Same as the `--no-connection` option.

- `connect-if-online`

  Use the database connection if possible, but continue using file system operations to copy the data files if a connection cannot be established. Same as the `--connect-if-online` option.

- `include`

  Specifies the regular expression to do a partial backup, including certain InnoDB tables only. Same as the `--include` option.

- `with-timestamp`

  Creates a subdirectory underneath the backup directory, with a name formed from the timestamp of the backup operation. Same as the `--with-timestamp` option.

- `slave-info`

  Assists in setting up a new slave instance using a backup of the master. Same as the `--slave-info` option. Same as that of existing innobackup option.

- `databases=`*`list`*

Space-separated list of non-InnoDB tables from selected databases to back up. Same as the `--databases` option.

- `databases-list-file=`*`path`*

  Specifies a file containing names of non-InnoDB tables from selected databases to back up. Same as the `--databases-list-file` option.

- `suspend-at-end`

  Pauses the backup so that you can code your own additional backup steps while the MySQL server is in a read-only state. Same as the `--suspend-at-end` option.

- `exec-when-locked="`*`utility arg1 arg2 ...`*`"`

  Specifies the command to run while the MySQL server is in a read-only state and the backup is suspended. Same as the `--exec-when-locked` option.

- `incremental`

  Performs an incremental backup. Same as the `--incremental` option.

- `incremental-with-redo-log-only`

  Performs an incremental backup. Same as the `--incremental-with-redo-log-only` option.

- `incremental-base`

  Specifies the location of a previous incremental backup, to automatically retrieve the LSN to use as the starting point for the next one. Same as the `--incremental-base` option.

- `start-lsn`

  Specifies the starting point for an incremental backup, in terms of a logical sequence number value. Same as the `--start-lsn` option.

- `only-known-file-types`

  Limits copying of non-InnoDB files to a specific set of file extensions. Same as the `--only-known-file-types` option.

- `limit-memory=`*`MB`*

  Specify maximum memory in megabytes that can be used in the `apply-log` operation. Same as the `--limit-memory` option.

- `sleep=`*`MS`*

  Specify the number in milliseconds to sleep after copying a certain amount data. Same as the `--sleep` option.

- `comments=`*`string`*

  Stores a user-specified string to identify the backup. Same as the `--comments` option.

- `comments-file=`*`path`*

  Stores a user-specified file to identify the backup. Same as the `--comments-file` option.

- `src-entry=`*`path`*

  Identifies a file or directory to extract from a single-file backup. Same as the `--src-entry` option.

- `dst-entry=`*`path`*

  Specifies the destination for the file or directory extracted from a single-file backup. Same as the `--dst-entry` option.

Chapter 5 Recovering or Restoring a Database

Table of Contents

The ultimate purpose of backup data is to help recover from a database issue, or to create a clone of the original database in another location (typically to run report queries or to create a new replication slave). This section describes the procedures to handle those various scenarios.

After a serious database issue, you might need to perform a recovery under severe time pressure. It is critical to confirm in advance:

- How long the recovery will take, including any steps to transfer, unpack, and otherwise process the data.
- That you have practiced and documented all steps of the recovery process, so that you can do it correctly in one try. If a hardware issue requires restoring the data to a different server, verify all privileges, storage capacity, and so on, on that server ahead of time.
- That you have periodically verified the accuracy and completeness of the backup data, so that the system will be up and running properly after being recovered.

5.1 Preparing the Backup to be Restored

Immediately after the backup job completes, the backup files might not be in a consistent state, because data could be inserted, updated, or deleted while the backup is running. These initial backup files are known as the raw backup.

You must update the backup files so that they reflect the state of the database corresponding to a specific InnoDB log sequence number. (The same kind of operation as crash recovery.) When this step is complete, these final files are known as the prepared backup.

During the backup, `mysqlbackup` copies the accumulated InnoDB log to a file called `ibbackup_logfile`. This log file is used to “roll forward” the backed-up data files, so that every page in the data files corresponds to the same log sequence number of the InnoDB log. This phase also creates new `ib_logfiles` that correspond to the data files.

The `mysqlbackup` option for turning a raw backup into a prepared backup is `apply-log`. You can run this step on the same database server where you did the backup, or transfer the raw backup files to a different system first, to limit the CPU and storage overhead on the database server.

> **Note**
>
> Since the `apply-log` operation does not modify any of the original files in the backup, nothing is lost if the operation fails for some reason (for example, insufficient disk space). After fixing the problem, you can safely retry `apply-log` and by specifying the `--force` [45] option, which allows the data and log files created by the failed `apply-log` operation to be overwritten.

For simple backups (without compression or incremental backup), you can combine the initial backup and the `apply-log` step using the option `backup-and-apply-log`.

Example 5.1 Applying the Log to a Backup

This example runs `mysqlbackup` to roll forward the data files so that the data is ready to be restored:

```
mysqlbackup --backup-dir=/export/backups/2011-06-21_8-36-58 apply-log
```

That command creates InnoDB log files (`ib_logfile*`) within the backup directory and applies log records to the InnoDB data files (`ibdata*` and `*.ibd`).

Example 5.2 Applying the Log to a Compressed Backup

If the backup is compressed, as in Section 3.3.3, “Making a Compressed Backup”, specify the `--uncompress` option to `mysqlbackup` when applying the log to the backup:

```
mysqlbackup --backup-dir=/export/backups/compressed --uncompress apply-log
```

Example 5.3 Applying an Incremental Backup to a Full Backup

After you take an incremental backup, as in Section 3.3.2, “Making an Incremental Backup”, the changes reflected in those backup files must be applied to a full backup to bring the full backup up-to-date, in the same way that you apply changes from the binary log.

To bring the data files from the full backup up to date, first run the apply log step so that the data files include any changes that occurred while the full backup was running. Then apply the changes from the incremental backup to the data files produced by the full backup:

```
mysqlbackup --backup-dir=/export/backups/full apply-log
mysqlbackup --backup-dir=/export/backups/full \
  --incremental-backup-dir=/export/backups/incremental \
  apply-incremental-backup
```

Now the data files in the `full-backup` directory are fully up-to-date, as of the time of the incremental backup.

5.2 Performing a Restore Operation

The `mysqlbackup` option to perform a restore operation is `copy-back`. The restoration process requires the database server to be already shut down. It copies the data files, logs, and other backed-up files from the backup directory back to their original locations, and performs any required post-processing on them. The options `datadir`, `innodb_log_files_in_group`, and `innodb_log_file_size` must be specified either in the target server's configuration file, in the file specified by the `--defaults-file` option, or as command-line options.

Example 5.4 Shutting Down and Restoring a Database

```
mysqladmin --defaults-file=/usr/local/mysql/my.cnf --user=root --password shutdown
mysqlbackup --defaults-file=/usr/local/mysql/my.cnf \
  --backup-dir=/export/backups/full \
    copy-back
```

> **Note**
>
> The restored data includes the `backup_history` table, where MySQL Enterprise Backup records details of each backup. Restoring this table to its earlier state removes information about any subsequent backups that you did. This is the correct starting point for future incremental backups, particularly those using the `--incremental-base` option.

Important

Before restoring a partial backup, you might need to delete first from the backup the `.frm` files associated with InnoDB tables that were not backed up. See Section 4.1.9, “Partial Backup Options” and Section 3.3.4.1, “Backing Up Some or All InnoDB Tables” for details.

5.3 Point-in-Time Recovery from a Hot Backup

Using MySQL Enterprise Backup on its own, you can restore your data as it was at certain moments in time: every N hours, every day at 2 AM, and so on depending on your backup schedule. To reproduce data based on an arbitrary time somewhere in between backup jobs, you can use MySQL Enterprise Backup in combination with the MySQL binary log feature.

To recover the database to a specific point in time:

- Binary logging must be enabled in MySQL, before taking the backup that serves as the base for this restore operation.
- Find the binlog position that corresponds to the time of the backup. InnoDB only stores the binlog position information to its tablespace at a transaction commit. **To make InnoDB aware of the current binlog position, you must run at least one transaction while binlogging is enabled.** When you run the `apply-log` operation on your backup, `mysqlbackup` prints the latest MySQL binlog position the backup knows of. Also, `mysqld` prints it when you start it on the restored data:

  ```
  $ mysqld --defaults-file=/export/mysql/my.cnf
  040122 15:41:57  InnoDB: Database was not shut down normally!
  InnoDB: Starting crash recovery.
  …
  InnoDB: Last MySQL binlog file position 0 27183537, file name ./binlog.000005
  …
  mysqld: ready for connections.
  ```

  The MySQL version must be >= 5.1.

  The printed position is the MySQL binlog byte position from the moment when MySQL Enterprise Backup finished copying your data files.

- Use the `mysqlbinlog` to dump all the SQL activity that happened after the binlog position of the backup. Direct the output of the `mysqlbinlog` command to an output file, instead of piping it directly to `mysql`. This output file contains timestamps for all SQL statements in the binlog.

  ```
  $ mysqlbinlog --start-position=27183537 /sqldata/binlog.000005 >partial_binlog
  $ mysqlbinlog /sqldata/binlog.000006 >> partial_binlog
  $ mysqlbinlog /sqldata/binlog.000007 >> partial_binlog
  ...
  $ mysqlbinlog /sqldata/binlog.00000N >> partial_binlog
  ```

- In an editor, remove all statements after the point in time to which you intend to restore.
- Replay the SQL statements to update the backup data with the same operations that happened after the backup. Process the modified file with `mysql`, like this:

  ```
  $ mysql < partial_binlog
  ```

- To recover the database to the latest possible time, skip the steps of saving the `mysqlbinlog` output in a file and removing recent SQL statements from it. Pipe the output from `mysqlbinlog --start-position=...` directly to `mysql` to replay all the SQL statements after the last backup.

For more tips on using the binary log for point-in-time recovery, see Point-in-Time (Incremental) Recovery Using the Binary Log.

5.4 Backing Up and Restoring a Single .ibd File

A table with a table-specific tablespace (stored in an .ibd file) can be restored individually without taking down the MySQL server. This technique is applicable if you delete or update the table data by mistake, without actually losing the table itself through a DROP TABLE, TRUNCATE TABLE, or DROP DATABASE statement.

If you have a clean backup of an .ibd file, you can restore it to the MySQL installation from which it originated as follows:

1. For MySQL 5.5 and earlier, the table must already exist and not have been dropped or truncated since taking the backup. When an InnoDB table is truncated, or dropped and recreated, it gets a new table ID. Any ID mismatch between the table in the database and the backed-up table can prevent it from being restored. The requirement for matching table IDs is also the reason why you must restore to the same MySQL server from which the backup data came, not another server with a similar set of databases and tables. This restriction does not apply to MySQL 5.6 and later, as long as the restoration is made from one Generally Available (GA) version to another in the same series of MySQL servers.

2. Prevent write operations for the table to be restored. This prevents users from modifying the table while the restore is in progress.

   ```
   LOCK TABLES tbl_name WRITE;
   ```

3. Issue this ALTER TABLE statement:

   ```
   ALTER TABLE tbl_name DISCARD TABLESPACE;
   ```

   Caution: This deletes the current .ibd file.

4. Copy the backup .ibd file back to the appropriate database directory.

5. Issue this ALTER TABLE statement:

   ```
   ALTER TABLE tbl_name IMPORT TABLESPACE;
   ```

6. Release the write lock to complete the restore procedure:

   ```
   UNLOCK TABLES;
   ```

In this context, a clean.ibd file backup means:

- There are no uncommitted modifications by transactions in the .ibd file.
- There are no unmerged insert buffer entries in the .ibd file.
- Purge has removed all delete-marked index records from the .ibd file.
- mysqld has flushed all modified pages of the .ibd file from the buffer pool to the file.

You can make such a clean backup .ibd file with the following method:

1. Stop all activity from the mysqld server and commit all transactions.

2. Wait until SHOW INNODB STATUS shows that there are no active transactions in the database, and the main thread status of InnoDB is Waiting for server activity. Then you can make a copy of the .ibd file.

Another method for making a clean copy of an .ibd file is to use mysqlbackup:

1. Use `mysqlbackup` with the `--only-innodb` or `--only-innodb-with-frm` option to back up the `InnoDB` installation.
2. Run `mysqlbackup ... apply-log` to create a consistent version of the backup database.
3. Start a second (dummy) `mysqld` server on the backup and let it clean up the `.ibd` files in the backup. Wait for the cleanup to end.
4. Shut down the dummy `mysqld` server.
5. Take a clean `.ibd` file from the backup.

5.5 Restoring a Backup with a Database Upgrade or Downgrade

> **Important**
>
> Due to the changes to the InnoDB storage engine going from MySQL 5.5 to 5.6, restoring a backup of a MySQL 5.5 database to a MySQL 5.6 server requires some extra steps beyond the general restore and upgrade procedures, the skipping of which will crash the target server. For such a restoration, follow the steps [73] described below.

You can back up a server running one MySQL version and restore on a server running a different MySQL version. After the restore, perform the appropriate upgrade steps as if you are running the new MySQL version for the first time. (Or, if you installed on a server running an older MySQL, perform the appropriate downgrade steps.) For information about upgrading and downgrading, see Upgrading MySQL and Downgrading MySQL.

> **Note**
>
> After upgrading between certain combinations of MySQL versions, you might see error messages about missing or mismatching definitions for system tables. Use the mysql_upgrade command as directed in the upgrade instructions to correct such issues. See `mysql_upgrade` — Check and Upgrade MySQL Tables for instructions on this command.

> **Warning**
>
> Restoring a database to an older MySQL version (i.e., server downgrading) is only supported when the original and the final versions are in the same release series (e.g. going from 5.5.30 to 5.5.29). Downgrading to a lower series (e.g. from 5.6.33 to 5.5.33) might cause server crashes or data corruption.

Steps to Back Up on MySQL 5.5 and Restore on MySQL 5.6

- Back up data on the MySQL 5.5 server.
- Install MySQL 5.6, without uninstalling MySQL 5.5.
- Restore data on the MySQL 5.6 server's data directory by simply running an `apply-log` and then a `copy-back` operation on the backup.
- Restart the old MySQL 5.5 server, using the data directory of the new MySQL 5.6 server as its own data directory.

> **Note**
>
> This is an extra step beyond the normal restore and upgrade procedures, special to the restoration of MySQL 5.5 data to MySQL 5.6 server; with it, the MySQL 5.5 server prepares the data for an upgrade to MySQL 5.6 by

> performing some clean-up steps on the data, similar to what the server would do during a crash recovery.

- Stop the MySQL 5.5 server.
- Start the new MySQL 5.6 server.
- Run upgrade steps as documented in the MySQL reference manual on the restored data. Make sure the mysql_upgrade that comes with MySQL 5.6 is applied.
- Check data.

Steps to Back Up on MySQL 5.1 and Restore on MySQL 5.5

- Back up on MySQL 5.1.
- Install MySQL 5.5.
- Restore on MySQL 5.5.
- Run upgrade steps as documented in the MySQL reference manual.
- Check data.

Chapter 6 Using MySQL Enterprise Backup with Replication

Table of Contents

Backup and restore operations are especially important in systems that use MySQL replication to synchronize data across a master server and a set of slave servers. In a replication configuration, MySQL Enterprise Backup helps you to deal with entire system images to set up new slave servers, or to restore a master server in an efficient way that avoids unnecessary work for the slave servers. Having multiple slave servers to choose from gives you more flexibility about where to perform backups. When the binary log is enabled, you have more flexibility about restoring to a point in time, even a time that is later than the last backup.

GTID support with MySQL Server 5.6 and above

MySQL Enterprise Backup supports the GTID feature of MySQL 5.6:

- The GTID feature is compatible with the CSV tables that the `mysqlbackup` command uses to log job progress and history.
- When a server using the GTID feature is backed up, `mysqlbackup` produces a file `gtid_executed.sql` as part of the backup data. This file contains a SQL statement that sets the `GTID_PURGED` configuration option. Execute this file using the `mysql` command line after restoring the backup data on a slave server. Optionally, you can uncomment the `CHANGE MASTER` command in this file and add any needed authentication parameters to it, before running it through `mysql`.
- For servers not using GTIDs, you can use the `--slave-info` option as before, then edit and execute the `ibbackup_slave_info` file afterward.

6.1 Setting Up a New Replication Slave

If you use MySQL replication, MySQL Enterprise Backup allows you to set up a slave database without stopping the master, by backing up the master and restoring that backup on a new slave server.

1. Take the backup, transfer it to the slave server, use `mysqlbackup` with the `apply-log` option to prepare it, and put the restored backup and the log files in the right directories for the new slave.
2. Edit the `my.cnf` file of the new slave and put `skip-slave-start` under the `[mysqld]` section.
3. Start the new slave `mysqld` (version >= 5.1). It prints the latest MySQL binlog position the backup knows of.

   ```
   …
   InnoDB: Last MySQL binlog file position 0 128760128, file name ./hundin-bin.006
   …
   ```

   Note that InnoDB only stores the binlog position information to its tablespace at a transaction commit. **To make InnoDB aware of the current binlog position, you must run at least one transaction while binlogging is enabled.**
4. Use the `CHANGE MASTER` SQL command on the slave to initialize it properly. For example:

   ```
   CHANGE MASTER TO
   MASTER_LOG_FILE='hundin-bin.006',
   MASTER_LOG_POS=128760128;
   ```

5. Set the statuses of any events that were copied from the master to `SLAVESIDE_DISABLED`. For example:

```
mysql> UPDATE TABLE mysql.event SET status = 'SLAVESIDE_DISABLED';
```

6. Start replication in the new slave with the `SLAVE START` SQL command.

7. Remove the line `skip-slave-start` from the `my.cnf` file of the slave.

6.2 Restoring a Master Database in Replication

To fix a corruption problem in a replication master database, you can restore the backup, taking care not to propagate unnecessary SQL operations to the slave servers:

1. Using the backup of the master database, do the `apply-log` operation, shut down the database, and do the `copy-back` operation.

2. Edit the master `my.cnf` file and comment out `log-bin`, so that the slaves do not receive twice the binlog needed to recover the master.

3. Replication in the slaves must be stopped temporarily while you pipe the binlog to the master. In the slaves, do:

```
mysql> STOP SLAVE;
```

4. Start the master `mysqld` on the restored backup:

```
$ mysqld
…
InnoDB: Doing recovery: scanned up to log sequence number 0 64300044
InnoDB: Last MySQL binlog file position 0 5585832, file name
./omnibook-bin.002
…
```

   InnoDB prints the binlog file and the position it was able to recover to.

5. Pipe the remaining binlog files to the restored backup. For example, if there are two more binlog files, `omnibook-bin.003` and `omnibook-bin,004`:

```
$ mysqlbinlog --start-position=5585832 mysqldatadir/omnibook-bin.002 | mysql
$ mysqlbinlog /mysqldatadir/omnibook-bin.003 | mysql
$ mysqlbinlog /mysqldatadir/omnibook-bin.004 | mysql
```

   The number of remaining binlog files varies depending on when the last backup was taken: the older the backup, the more remaining binlog files there may be.

6. The master database is now recovered. Shut down the master and edit `my.cnf` to uncomment `log-bin`.

7. Start the master again.

8. Start replication in the slaves again:

```
mysql> START SLAVE;
```

Chapter 7 Performance Considerations for MySQL Enterprise Backup

Table of Contents

7.1 Optimizing Backup Performance

This section describes the performance considerations for backup operations with the MySQL Enterprise Backup product. When optimizing and tuning the backup procedure, measure both the raw performance (how long it takes the backup to complete) and the amount of overhead on the database server. When measuring backup performance, consider:

- The limits imposed by your backup procedures. For example, if you take a backup every 8 hours, the backup must take less than 8 hours to finish.
- The limits imposed by your network and storage infrastructure. For example, if you need to fit many backups on a particular storage device, you might use compressed backups, even if that made the backup process slower.
- The tradeoff between backup time and restore time. You might choose a set of options resulting in a slightly slower backup, if those options enable the restore to be much faster. See Section 7.2, “Optimizing Restore Performance” for performance information for the restore process.

Full or Incremental Backup

After taking a full backup, subsequent backups can be performed more quickly by doing incremental backups, where only the changed data is backed up. For an incremental backup, specify the `--incremental` or `--incremental-with-redo-log-only` option to `mysqlbackup`. See Section 4.1.8, “Incremental Backup Options” for information about these options. For usage instructions for the backup and apply stages of incremental backups, see Section 3.3.2, “Making an Incremental Backup” and Example 5.3, “Applying an Incremental Backup to a Full Backup”.

Compressed Backup

Compressing the backup data before transmitting it to another server involves additional CPU overhead on the database server where the backup takes place, but less network traffic and less disk I/O on the server that is the final destination for the backup data. Consider the load on your database server, the bandwidth of your network, and the relative capacities of the database and destination servers when deciding whether or not to use compression. See Section 3.3.3, “Making a Compressed Backup” and Section 4.1.7, “Compression Options” for information about creating compressed backups.

Compression involves a tradeoff between backup performance and restore performance. In an emergency, the time needed to uncompress the backup data before restoring it might be unacceptable. There might also be storage issues if there is not enough free space on the database server to hold both the compressed backup and the uncompressed data. Thus, the more critical the data is, the more likely that you might choose not to use compression: accepting a slower, larger backup to ensure that the restore process is as fast and reliable as possible.

Single-File Backups

The single-file backup by itself is not necessarily faster than the traditional type of backup that produces a directory tree of output files. Its performance advantage comes from combining different

steps that you might otherwise have to perform in sequence, such as combining the backup data into a single output file and transferring it to another server. See Section 4.1.1.4, “Work with Single-File Backups” for the options related to single-file backups, and Section 3.3.5, “Making a Single-File Backup” for usage instructions.

InnoDB Configuration Options Settings

Prior to MySQL 5.5, it was common practice to keep the redo logs fairly small to avoid long startup times when the MySQL server was killed rather than shut down normally. In MySQL 5.5 and higher, the performance of crash recovery is significantly improved. With those releases, you can make your redo log files bigger if that helps your backup strategy and your database workload.

As discussed later, there are a number of reasons why you might prefer to run with the setting `innodb_file_per_table=1`.

Parallel Backup

The `mysqlbackup` command can take advantage of modern multicore CPUs and operating system threads to perform backup operations in parallel. See Section 4.1.11, “Performance / Scalability / Capacity Options” for the options to control how many threads are used for different aspects of the backup process. If you see that there is unused system capacity during backups, consider increasing the values for these options and testing whether doing so increases backup performance:

- When tuning and testing backup performance using a RAID storage configuration, consider the combination of option settings `--read-threads=3 --process-threads=6 --write-threads=3`. Compare against the combination `--read-threads=1 --process-threads=6 --write-threads=1`.
- When tuning and testing backup performance using a non-RAID storage configuration, consider the combination of option settings `--read-threads=1 --process-threads=6 --write-threads=1`.
- When you increase the values for any of the 3 “threads” options, also increase the value of the `--limit-memory` option, to give the extra threads enough memory to do their work.
- If the CPU is not too busy (less than 80% CPU utilization), increase the value of the `--process-threads` option.
- If the storage device that you are backing up from (the source drive) can handle more I/O requests, increase the value of the `--read-threads` option.
- If the storage device that you are backing up to (the destination drive) can handle more I/O requests, increase the value of the `--write-threads` option.

Depending on your operating system, you can measure resource utilization using commands such as `top`, `iostat`, `sar`, `dtrace`, or a graphical performance monitor. Do not increase the number of read or write threads `iowait` once the system `iowait` value reaches approximately 20%.

MyISAM Considerations

IMPORTANT:

- Although the `mysqlbackup` command backs up InnoDB tables without interrupting database use, the final stage that copies non-InnoDB files (such as MyISAM tables and `.frm` files) temporarily puts the database into a read-only state, using the statement `FLUSH TABLES WITH READ LOCK`. For best backup performance and minimal impact on database processing:

  1. Do not run long `SELECT` queries or other SQL statements at the time of the backup run.

2. Keep your MyISAM tables relatively small and primarily for read-only or read-mostly work.

Then the locked phase at the end of a `mysqlbackup` run is short (maybe a few seconds), and does not disturb the normal processing of `mysqld` much. If the preceding conditions are not met in your database application, use the `--only-innodb` or `--only-innodb-with-frm` option to back up only InnoDB tables, or use the `--no-locking` option to back up non-InnoDB files. Note that MyISAM, `.frm`, and other files copied under the `--no-locking` setting cannot be guaranteed to be consistent, if they are updated during this final phase of the backup.

- For a large database, a backup run might take a long time. Always check that `mysqlbackup` has completed successfully, either by verifying that the `mysqlbackup` command returned exit code 0, or by observing that `mysqlbackup` has printed the text “mysqlbackup completed OK!”.
- The `mysqlbackup` command is not the same as the former “MySQL Backup” open source project from the MySQL 6.0 source tree. The MySQL Enterprise Backup product supersedes the MySQL Backup initiative.
- Schedule backups during periods when no DDL operations involving tables are running. See Section A.1, “Limitations of `mysqlbackup` Command” for restrictions on backups at the same time as DDL operations.

Network Performance

For data processing operations, you might know the conventional advice that Unix sockets are faster than TCP/IP for communicating with the database. Although the `mysqlbackup` command supports the options `--protocol=tcp`, `--protocol=socket`, and `--protocol=pipe`, these options do not have a significant effect on backup or restore performance. These processes involve file-copy operations rather than client/server network traffic. The database communication controlled by the `--protocol` option is low-volume. For example, `mysqlbackup` retrieves information about database parameters through the database connection, but not table or index data.

Data Size

If certain tables or databases contain non-critical information, or are rarely updated, you can leave them out of your most frequent backups and back them up on a less frequent schedule. See Section 4.1.9, “Partial Backup Options” for information about the relevant options, and Section 3.3.4, “Making a Partial Backup” for instructions about leaving out data from specific tables, databases, or storage engines. Partial backups are faster because they copy, compress, and transmit a smaller volume of data.

To minimize the overall size of `InnoDB` data files, consider enabling the MySQL configuration option `innodb_file_per_table`. This option can minimize data size for `InnoDB` tables in several ways:

- It prevents the `InnoDB` system tablespace from ballooning in size, allocating disk space that can afterwards only be used by MySQL. For example, sometimes huge amounts of data are only needed temporarily, or are loaded by mistake or during experimentation. Without the `innodb_file_per_table` option, the system tablespace expands to hold all this data, and never shrinks afterward.
- It immediately frees the disk space taken up by an `InnoDB` table and its indexes when the table is dropped or truncated. Each table and its associated indexes are represented by a .ibd file that is deleted or emptied by these DDL operations.
- It allows unused space within a `.ibd` file to be reclaimed by the `OPTIMIZE TABLE` statement, when substantial amounts of data are removed or indexes are dropped.

- It enables partial backups where you back up some `InnoDB` tables and not others, as discussed in Section 3.3.4, “Making a Partial Backup”.

Avoid creating indexes that are not used by queries. Because indexes take up space in the backup data, unnecessary indexes slow down the backup process. (The copying and scanning mechanisms used by `mysqlbackup` do not rely on indexes to do their work.) For example, it is typically not helpful to create an index on each column of a table, because only one index is used by any query. Because the primary key columns are included in each `InnoDB` secondary index, it wastes space to define primary keys composed of numerous or lengthy columns, or multiple secondary indexes with different permutations of the same columns.

The Apply-Log Phase

If you store the backup data on a separate machine, and that machine is not as busy the machine hosting the database server, you can offload some postprocessing work (the apply-log phase) to that separate machine. Section 4.1.1.2, “Apply-Log Operations for Existing Backup Data”

There is always a performance tradeoff between doing the apply-log phase immediately after the initial backup (makes restore faster), or postponing it until right before the restore (makes backup faster). In an emergency, restore performance is the most important consideration. Thus, the more crucial the data is, the more important it is to run the apply-log phase immediately after the backup. Either combine the backup and apply-log phases on the same server by specifying the `backup-and-apply-log` option, or perform the fast initial backup, transfer the backup data to another server, and then perform the apply-log phase using one of the options from Section 4.1.1.2, “Apply-Log Operations for Existing Backup Data”.

7.2 Optimizing Restore Performance

This section describes the performance considerations for restore operations with the MySQL Enterprise Backup product. This subject is important enough to deserve its own section, separate from the discussion of backup performance, because:

- The restore operation is the phase of the backup-restore cycle that tends to vary substantially between different backup methods. For example, backup performance might be acceptable using `mysqldump`, but `mysqldump` typically takes much longer than MySQL Enterprise Backup for a restore operation.
- The restore operation is often performed during an emergency, where it is critical to minimize the downtime of the application or web site.
- The restore operation is always performed with the database server shut down.
- The restore operation is mainly dependent on low-level considerations, such as I/O and network speed for transferring files, and CPU speed, processor cores, and so on for uncompressing data.

For the combination of options you can specify for a restore job, see Section 4.1.1.3, “Restore an Existing Backup”.

Restoring Different Classes of Backup Data

Restoring a partial backup takes less time than restoring a full backup, because there is less data to physically copy. See Section 4.1.9, “Partial Backup Options” for information about making partial backups.

Restoring a compressed backup takes more time than restoring an uncompressed backup, because the time needed to uncompress the data is typically greater than any time saved by transferring less data across the network. If you need to rearrange your storage to free up enough space to uncompress the backup before restoring it, include that administration work in your estimate of the total time required. In an emergency, the time needed to uncompress the backup data before restoring it might be unacceptable. on the database server to hold both the compressed backup and the uncompressed

data. Thus, the more critical the data is, the more likely that you might choose not to use compression: accepting a slower, larger backup to ensure that the restore process is as fast and reliable as possible. See Section 4.1.7, “Compression Options” for information about making compressed backups.

The unpacking process to restore a single-file backup is typically not expensive either in terms of raw speed or extra storage. Each file is unpacked directly to its final destination, the same as if it was copied individually. Thus, if you can speed up the backup substantially or decrease its storage requirements by using single-file backups, that typically does not involve a tradeoff with restore time. See Section 4.1.1.4, “Work with Single-File Backups” for information about making single-file backups.

The Apply-Log Phase

If you store the backup data on a separate machine, and that machine is not as busy the machine hosting the database server, you can offload some postprocessing work (the apply-log phase) to that separate machine. Section 4.1.1.2, “Apply-Log Operations for Existing Backup Data”

There is always a performance tradeoff between doing the apply-log phase immediately after the initial backup (makes restore faster), or postponing it until right before the restore (makes backup faster). In an emergency, restore performance is the most important consideration. Thus, the more crucial the data is, the more important it is to run the apply-log phase immediately after the backup. Either combine the backup and apply-log phases on the same server by specifying the `backup-and-apply-log` option, or perform the fast initial backup, transfer the backup data to another server, and then perform the apply-log phase using one of the options from Section 4.1.1.2, “Apply-Log Operations for Existing Backup Data”.

Section 4.1.1.2, “Apply-Log Operations for Existing Backup Data”

Network Performance

For data processing operations, you might know the conventional advice that Unix sockets are faster than TCP/IP for communicating with the database. Although the `mysqlbackup` command supports the options `--protocol=tcp`, `--protocol=socket`, and `--protocol=pipe`, these options do not have a significant effect on backup or restore performance. These processes involve file-copy operations rather than client/server network traffic. The database communication controlled by the `--protocol` option is low-volume. For example, `mysqlbackup` retrieves information about database parameters through the database connection, but not table or index data.

Parallel Restore

The `mysqlbackup` command can take advantage of modern multicore CPUs and operating system threads to perform backup operations in parallel. See Section 4.1.11, “Performance / Scalability / Capacity Options” for the options to control how many threads are used for different aspects of the restore process. If you see that there is unused system capacity during a restore, consider increasing the values for these options and testing whether doing so increases restore performance:

- When tuning and testing backup performance using a RAID storage configuration, consider the combination of option settings `--read-threads=3 --process-threads=6 --write-threads=3`. Compare against the combination `--read-threads=1 --process-threads=6 --write-threads=1`.
- When tuning and testing backup performance using a non-RAID storage configuration, consider the combination of option settings `--read-threads=1 --process-threads=6 --write-threads=1`.
- When you increase the values for any of the 3 “threads” options, also increase the value of the `--limit-memory` option, to give the extra threads enough memory to do their work.
- If the CPU is not too busy (less than 80% CPU utilization), increase the value of the `--process-threads` option.

- If the storage device that you are restoring from (the source drive) can handle more I/O requests, increase the value of the `--read-threads` option.
- If the storage device that you are restoring to (the destination drive) can handle more I/O requests, increase the value of the `--write-threads` option.

Depending on your operating system, you can measure resource utilization using commands such as `top`, `iostat`, `sar`, `dtrace`, or a graphical performance monitor. Do not increase the number of read or write threads `iowait` once the system `iowait` value reaches approximately 20%.

Chapter 8 Using MySQL Enterprise Backup with Media Management Software (MMS) Products

Table of Contents

This section describes how you can use MySQL Enterprise Backup in combination with media management software (MMS) products. Such products are typically used for managing large volumes of backup data, often with high-capacity backup devices such as tape drives.

8.1 Backing Up to Tape with Oracle Secure Backup

Tape drives are affordable, high-capacity storage devices for backup data. The MySQL Enterprise Backup product can interface with media management software (MMS) such as Oracle Secure Backup (OSB) to drive MySQL backup and restore jobs. The media management software must support Version 2 or higher of the System Backup to Tape (SBT) interface.

On the MySQL Enterprise Backup side, you run the backup job as a single-file backup using the `--backup-image` parameter, with the prefix `sbt:` in front of the filename, and optionally pass other `--sbt-*` parameters to the `mysqlbackup` command to control various aspects of the SBT processing. The `--sbt-*` options are listed in Section 4.1.10, “Single-File Backup Options”.

On the OSB side, you can schedule MySQL Enterprise Backup jobs by specifying a configurable command that calls `mysqlbackup`. You control OSB features such as encryption by defining a “storage selector” that applies those features to a particular backup, and passing the name of the storage selector to OSB using the MySQL Enterprise Backup parameter `--sbt-database-name=`*`storage_selector`*.

To back up MySQL data to tape:

- Specify the `--backup-image=sbt:`*`name`* parameter of the `mysqlbackup` command to uniquely identify the backup data. The `sbt:` prefix sends the backup data to the MMS rather than a local file, and the remainder of the argument value is used as the unique backup name within the MMS.
- Specify the `--sbt-database-name` parameter of the `mysqlbackup` command to enable the OSB operator to configure a storage selector for backups from this MySQL source. (This parameter refers to a “storage selector” defined by the OSB operator, not to any MySQL database name.) By default, `mysqlbackup` supplies a value of `MySQL` for this MMS parameter. The argument to this option is limited to 8 bytes.
- If you have multiple media management programs installed, to select the specific SBT library to use, specify the `--sbt-lib-path` parameter of the `mysqlbackup` command. If you do not specify the `--sbt-lib-path parameter`, `mysqlbackup` uses the normal operating system paths and environment variables to locate the SBT library, which is named `libobk.so` on Linux and Unix systems and `ORASBT.DLL` on Windows systems. When you specify `--sbt-lib-path`, you can use a different filename for the library in addition to specifying the path.
- Specify any other product-specific settings that are normally controlled by environment variables using the `--sbt-environment` option.

To restore MySQL data from tape:

- Specify the `--backup-image=sbt:`*`name`* parameter of the `mysqlbackup` command as part of the restore operation. Use the same *`name`* value as during the original backup. This single parameter retrieves the appropriate data from the appropriate tape device.

- Optionally use the `--sbt-lib-path` option, using the same values as for the backup operation.
- Specify any other product-specific settings that are normally controlled by environment variables using the `--sbt-environment` option.

For product-specific information about Oracle Secure Backup, see the Oracle Secure Backup documentation.

Example 8.1 Sample `mysqlbackup` Commands Using MySQL Enterprise Backup with Oracle Secure Backup

```
# Uses libobk.so or ORASBT.DLL in standard places):
mysqlbackup --port=3306 --protocol=tcp --user=root --password \
  --backup-image=sbt:backup-shoeprod-2011-05-30 \
  --backup-dir=/backup backup-to-image

# Associates this backup with storage selector 'shoeprod':
mysqlbackup --port=3306 --protocol=tcp --user=root --password \
  --backup-image=sbt:backup-shoeprod-2011-05-30 \
  --sbt-database-name=shoeprod \
  --backup-dir=/backup backup-to-image

# Uses an alternative SBT library, /opt/Other-MMS.so:
mysqlbackup --port=3306 --protocol=tcp --user=root --password \
  --backup-image=sbt:backup-shoeprod-2011-05-30 \
  --sbt-lib-path=/opt/Other-MMS.so \
  --backup-dir=/backup backup-to-image
```

Chapter 9 Troubleshooting for MySQL Enterprise Backup

Table of Contents

To troubleshoot issues regarding backup and restore with the MySQL Enterprise Backup product, consider the following aspects:

- Before troubleshooting any problem, familiarize yourself with the known limits and restrictions on the product, in Appendix A, *MySQL Enterprise Backup Limitations*.
- If the `mysqlbackup` command encounters problems during operating system calls, it returns the corresponding OS error codes. You might need to consult your operating system documentation for the meaning and solution of these error codes.
- The output from the `mysqlbackup` command is sent to `stderr` rather than `stdout`. Use appropriate redirection, such as `2>log_file`, to capture backup output for use in error diagnosis.
- Incremental backups require care to specify a sequence of time periods You must record the final LSN value at the end of each backup, and specify that value in the next incremental backup. You must also make sure that the full backup you restore is prepared correctly first, so that it contains all the changes from the sequence of incremental backups.
- As the `mysqlbackup` command proceeds, it writes progress information into the `mysql.backup_progress` table. When the command finishes the backup operation, it records status information in the `mysql.backup_history` table. You can query these tables to monitor ongoing jobs, see how much time was needed for various stages, and check if any errors occurred.

9.1 Monitoring Backups with MySQL Enterprise Monitor

With the combination of the MySQL Enterprise Backup and MySQL Enterprise Monitor products, you can monitor the progress and history of backup jobs without writing your own queries or scripts:

- The MySQL Enterprise Monitor graphs `Backup Run Time` and `Backup Locked Time` chart how long the phases of backup jobs take.
- The MySQL Enterprise Monitor rules `MySQL Enterprise Backup Failed`, `MySQL Enterprise Backup Succeeded`, `MySQL Enterprise Backup Lock Time Excessive`, `Incremental MySQL Enterprise Backups Not Enabled`, and `Last Full MySQL Enterprise Backup Too Old` alert you to issues related to backup jobs.

The monitoring capability requires MySQL Enterprise Backup 3.5.3 and higher, and MySQL Enterprise Monitor 2.3.4 and higher. For information about these MySQL Enterprise Monitor features, see the MySQL Enterprise Monitor User's Guide.

9.2 Error codes of MySQL Enterprise Backup

The return code of the MySQL Enterprise Backup (`mysqlbackup`) process is 0 if the backup or restore run succeeds. If the run fails for any reason, the return code is 1.

9.3 Working Around Corruption Problems

Sometimes the operating system or the hardware can corrupt a data file page, in a location that does not cause a database error but prevents `mysqlbackup` from completing:

```
mysqlbackup: Re-reading page at offset 0 3185082368 in /sqldata/mts/ibdata15
bbackup: Re-reading page at offset 0 3185082368 in /sqldata/mts/ibdata15
bbackup: Error: page at offset 0 3185082368 in /sqldata/mts/ibdata15 seems corrupt!
```

A corruption problem can have different causes. Here are some suggestions for dealing with it:

- The problem can occur if the MySQL server is too busy. Before trying other solutions, you might want to perform the backup again using some non-default settings for the following `mysqlbackup` options:
  - `--page-reread-time=`*`MS`*. Try set the value to, for example, “0.05”, for faster rereads during checksum failures.
  - `--page-reread-count=`*`retry_limit`*. Try set the value to, for example, “1000”, to allow more rereads during checksum failures before MySQL Enterprise Backup gives up and throws an error.
- Scrambled data in memory can cause the problem even though the data on disk is actually uncorrupted. Reboot the database server and the storage device to see if the problem persists.
- If the problem persists after the database server and the storage device have been restarted, you might really have a corruption on your disk. You might consider restoring data from an earlier backup and "roll forward" the recent changes to bring the database back to its current state.
- If you want to make MySQL Enterprise Backup finish a backup anyway before you go and investigate the root cause of the issue, you can rewrite the checksum values on the disk by running the innochecksum utility on the server:

  ```
  innochecksum --no-checksum --write=crc32
  ```

  The option `--no-checksum` disable the verification function of the tool, and the option `--write=crc32` makes innochecksum rewrite the checksum values on the disk.

IMPORTANT: Do not treat corruption problems as a minor annoyance. Find out what is wrong with the system that causes the corruption—however, such troubleshooting is beyond the scope of this manual.

9.4 Using the MySQL Enterprise Backup Logs

The `mysql.backup_progress` table lets you monitor backup jobs as they run. The `mysql.backup_history` table lets you see the results of completed jobs. Because these tables are created with the CSV storage engine, you can query them from SQL, or parse the text files from an application or script.

To skip updating these tables for a backup operation, use the `--no-history-logging` option.

`backup_progress` Table

Each row in the `backup_progress` table records a state change or message from a running backup job. The `backup_progress` table has the following columns:

- `backup_id`
- `tool_name`
- `error_code`
- `error_message`
- `current_time`

- current_state

Because the CSV storage engine cannot represent NULL values directly, the logs use a -1 value instead, for example in the binlog_pos column if binary logging is not enabled.

Use the backup_id value to group together the information for a single backup operation, and to correlate with the corresponding row in the backup_history table after the job is finished.

Use the error_code and error_message values to track the progress of the job, and detect if a serious error occurs that requires stopping the backup operation.

Use the current_time and current_state values to measure how long each part of the backup operation takes, to help with planning the time intervals for future backups.

backup_history Table

Each row in the backup_history table records the details of one completed backup job, produced by the mysqlbackup command. The backup_history table has the following columns:

- backup_id
- tool_name
- start_time
- end_time
- binlog_pos
- binlog_file
- compression_level
- engines
- innodb_data_file_path
- innodb_file_format
- start_lsn
- end_lsn
- backup_type
- backup_format
- mysql_data_dir
- innodb_data_home_dir
- innodb_log_group_home_dir
- innodb_log_files_in_group
- innodb_log_file_size
- backup_destination
- lock_time
- exit_state
- last_error

- `last_error_code`

Use the `end_lsn` value to automate operations related to incremental backups. When you take a full or incremental backup, you specify the end LSN from that backup as the starting LSN for the next incremental backup.

Use the values that correspond to backup-related configuration settings, such as `mysql_data_dir`, `innodb_data_home_dir`, and `backup_destination`, to confirm that the backups are using the right source and destination directories.

Use the values `exit_state`, `last_error`, and `last_error_code` to evaluate the success or failure of each backup.

If `last_error` is `'NO_ERROR'`, the backup operation was successful. In case of any errors, you can retrieve the full list of errors for that backup operation from the `backup_progress` table.

9.5 Using the MySQL Enterprise Backup Manifest

Each backup directory includes some files in the `meta` subdirectory that detail how the backup was produced, and what files it contains. The files containing this information are known collectively as the manifest.

`mysqlbackup` produces these files for use by database management tools; it does not consult or modify the manifest files after creating them. Management tools can use the manifest during diagnosis and troubleshooting procedures, for example where the original MySQL instance has been lost entirely and the recovery process is more substantial than copying files back to a working MySQL server.

The files in the manifest include:

- `backup_create.xml`: information about the backup operation.
- `backup_content.xml`: information about the files in the backup. This information is only complete and consistent when the backup operation succeeds. The contents of this file might be expanded in the future. A management tool might use this information to confirm which tables are part of a full backup, or a partial backup performed with the `--databases` option. (The information is not present for partial backups taken with the `--include`, `--incremental`, `--incremental-with-redo-log-only`, `--only-innodb`, or `--only-innodb-with-frm` options.) A management tool might compare the checksum recorded in the manifest for a single-file backup against the checksum for the file after the single-file backup is unpacked.
- `image_files.xml`: information about the files in a single-file backup. (Only produced for backups taken with the `backup-to-image` and `backup-dir-to-image` options.) A management tool might use the paths recorded in this file to plan or automate the unpacking of a single-file backup using the `image-to-backup-dir` or `extract` options, or to remap the paths of extracted files with the `--src-entry` and `--dst-entry` options.

Chapter 10 Frequently Asked Questions for MySQL Enterprise Backup

This section lists some common questions about MySQL Enterprise Backup, with answers and pointers to further information.

Questions

Questions and Answers

10.1: Does MySQL Enterprise Backup work with MySQL Server version *x*.*y*.*z*?

See Section B.1, “File Compatibility with Older MySQL or InnoDB Versions” for details of compatibility between different releases of MySQL Enterprise Backup and MySQL Server.

10.2: What is the big `ibdata` file that is in all the backups?

You might find your backup data taking more space than expected because of a large file with a name such as `ibdata1`. This file represents the InnoDB system tablespace, which grows but never shrinks, and is included in every full and incremental backup. To reduce the space taken up by this file in your backup data:

- After doing a full backup, do a succession of incremental backups, which take up less space. The `ibdata1` file in the incremental backups is typically much smaller, containing only the portions of the system tablespace that changed since the full backup.
- Set the configuration option `innodb_file_per_table=1` before creating your biggest or most active InnoDB tables. Those tables are split off from the system tablespaces into separate `.ibd` files, which are more flexible in terms of freeing disk space when dropped or truncated, and can be individually included or excluded from backups.
- If your system tablespace is very large because you created a high volume of InnoDB data before turning on the `innodb_file_per_table` setting, you might use `mysqldump` to dump the entire instance, then turn on `innodb_file_per_table` before re-creating it, so that all the table data is kept outside the system tablespace.

10.3: Can I back up non-InnoDB data with MySQL Enterprise Backup?

While MySQL Enterprise Backup can back up non-InnoDB data (like MYISAM tables), the MySQL server to be backed up must support InnoDB (i.e., the backup process will fail if the server was started up with the `--innodb=OFF` or `--skip-innodb` option), and the server must contain at least one InnoDB table.

10.4: What happens if “apply” step is interrupted?

If the `mysqlbackup` command is interrupted during the `apply-log` or `apply-incremental-backup` stage, the backup data is OK. The file operations performed by those options can be

performed multiple times without harming the consistency of the backup data. Just run the same `mysqlbackup` command again, and when it completes successfully, all the necessary changes are present in the backup data.

10.5: Why is the option `--defaults-file` not recognized?

When you specify the `--defaults-file` option, it must be the first option after the name of the command. Otherwise, the error message makes it look as if the option name is not recognized.

10.6: Can I back up a database on one OS platform and restore it on another one using MySQL Enterprise Backup?

See Section B.9, “Cross-Platform Compatibility” for details.

Part III Appendixes

Table of Contents

Appendix A MySQL Enterprise Backup Limitations

Table of Contents

Please refer to the MySQL Enterprise Backup version history in Appendix D, *MySQL Enterprise Backup Change History* for a list of fixed `mysqlbackup` bugs.

A.1 Limitations of `mysqlbackup` Command

- The group commit feature in MySQL 5.6 and higher changes the frequency of flush operations for the `InnoDB` redo log, which could affect the point in time associated with the backup data from `InnoDB` tables. See Section B.2, “Compatibility Notes for MySQL Versions” for details.
- When restoring an individual InnoDB table, as described in Section 5.4, “Backing Up and Restoring a Single `.ibd` File”, the table must not have been dropped or truncated in the MySQL server after the backup. Dropping or truncating an InnoDB table changes its internal table ID, and when the table is re-created the ID will not match the table ID from the backup data.
- In Linux, Unix, and OS X systems, the `mysqlbackup` command does not record file ownership or permissions of the files that are backed up. Upon restore, these files might have different ownership, for example being owned by `root` rather than `mysql`. They might also have different read/write permissions, for example being readable by anyone rather than just the file owner. When planning your backup strategy, survey the files in the MySQL data directory to ensure they have consistent owner and permission settings. When executing a restore operation, use an appropriate combination of `su`, `umask`, `chown`, and `chmod` on the restored files to set up the same owners and privileges as on the original files.
- In some cases, backups of non-transactional tables such as `MyISAM` tables could contain additional uncommitted data. If autocommit is turned off, and both `InnoDB` tables and non-transactional tables are modified within the same transaction, data can be written to the non-transactional table before the binlog position is updated. The binlog position is updated when the transaction is committed, but the non-transactional data is written immediately. If the backup occurs while such a transaction is open, the backup data contains the updates made to the non-transactional table.
- If the `mysqlbackup` process is interrupted, such as by a Unix `kill -9` command, a `FLUSH TABLES WITH READ LOCK` operation might remain running. In this case, use the `KILL QUERY` statement from the `mysql` command line to kill the `FLUSH TABLES WITH READ LOCK` statement. This issue is more likely to occur if the `FLUSH TABLES` operation is stalled by a long-running query or transaction. Refer to Chapter 4, *`mysqlbackup` Command Reference* for guidelines about backup timing and performance.
- Do not run the DDL operations `ALTER TABLE`, `TRUNCATE TABLE`, `OPTIMIZE TABLE`, `REPAIR TABLE`, or `RESTORE TABLE` while a backup operation is going on. The resulting backup might be corrupted.

  The only `ALTER TABLE` operations that can be safely run in parallel with a backup are those that do not influence the physical representation of records on disk, such as changing column names or default column values.
- The maximum number of subdirectories allowed in the `--backup-dir` path is 21. This limit could be exceeded by a deeply nested backup directory, or by an anomalous condition such as symbolic links forming an infinite recursive path.
- If you take a backup when there are temporary tables in the database, and you use those temporary tables to update or insert into normal tables, then applying the MySQL binlog to a backup can fail.

That is, you might not be able to roll forward the backup to a particular point in time using the MySQL binlog. Temporary tables are not copied to the backup because the physical filenames `#sql*.frm` do not correspond to the logical table names that MySQL writes to the binlog. This problem might be removed in the future, if MySQL implements “row-level binlogging”.

- Currently, if the regular expression for the `--include` option does not match any table names, *all* file-per-table tables are included in the backup.
- The `engines` column in the `mysql.backup_history` table does not correctly reflect the storage engines of the backed-up databases.
- `apply-log` might fail with an assertion error when applied on backups created for MySQL Server 5.6 with `GTID_MODE=ON` and if there were high DML traffic and online DDL operations during the backup operations.

Appendix B Compatibility Information for MySQL Enterprise Backup

Table of Contents

This section describes information related to compatibility issues for MySQL Enterprise Backup releases, including useful information for users migrating from the `innobackup` and `ibbackup` commands available in MySQL Enterprise Backup 3.5 and the InnoDB Hot Backup products.

> **Note**
>
> Starting in MySQL Enterprise Backup 3.7.1, the `innobackup` and `ibbackup` commands that are aliases of the `mysqlbackup` display warning messages when you use them.

B.1 File Compatibility with Older MySQL or InnoDB Versions

Each release of MySQL Enterprise Backup can restore backups produced by older MySQL Enterprise Backup releases.

From time to time, changes are made to the format of MySQL data and log files. These changes can make older MySQL Enterprise Backup versions incompatible with the new MySQL version.

Currently, these are the major MySQL/InnoDB versions: 3.23 (first released in May 12, 2001), 4.0 (December 23, 2001), 4.1 (April 3, 2003), 5.0 (December 24, 2003), 5.1 (November 29, 2005), 5.5 (December 15, 2010), and 5.6 (February 4, 2013).

Versions of MySQL Enterprise Backup from 3.5 to 3.8 are all compatible with MySQL/InnoDB version 5.1 and up.

Versions of MySQL Enterprise Backup from 3.5 to 3.7 are compatible with MySQL/InnoDB version 5.0 and up.

IMPORTANT: Backing up tables using the Barracuda file format, which is available with the combination of MySQL and the InnoDB Plugin, requires MySQL Enterprise Backup 3.5 or newer.

For MySQL versions prior to 5.0, the corresponding backup product was the InnoDB Hot Backup product, which was the ancestor of MySQL Enterprise Backup. InnoDB Hot Backup continues to be compatible with MySQL 5.5, 5.1, and 5.0, with the exception of InnoDB tables in the Barracuda file format; to use InnoDB Hot Backup, all InnoDB tables must use the Antelope file format.

B.2 Compatibility Notes for MySQL Versions

This section lists any performance-related features and settings in MySQL Server versions that affect various aspects of the backup process.

MySQL 5.6

Some new MySQL 5.6 features introduce changes in directory layout and file contents for InnoDB tables. Backing up servers that use these features requires MySQL Enterprise Backup 3.8.1 or higher:

- `innodb_page_size` configuration option.
- `innodb_undo_directory`, `innodb_undo_logs`, and `innodb_undo_tablespaces` configuration options.
- `innodb_checksum_algorithm` configuration option.
- `DATA DIRECTORY` clause of the `CREATE TABLE` statement, which produces a .isl file in the database directory and stores the .ibd file in a user-specified location.
- Online DDL.

The group commit feature in MySQL 5.6 reduces the frequency of flush operations for the `InnoDB` redo log, relying more on the binary log to ensure consistency. Because MySQL Enterprise Backup does not back up or replay transactions from the binary log, the data from `InnoDB` tables might correspond to a time during the backup, before the `FLUSH TABLES WITH READ LOCK` phase, while the data from non-`InnoDB` tables corresponds to the time when the `FLUSH TABLES WITH READ LOCK` phase starts.

See Section D.2, “Changes in MySQL Enterprise Backup 3.8.1 (2013-02-05)” for details on the fixes and enhancements related to these MySQL 5.6 features.

B.3 Compatibility of Backup Data with Other MySQL Enterprise Backup Versions

Backups produced with any 3.x version of MySQL Enterprise Backup can be restored using any higher MySQL Enterprise Backup version.

Backups produced with MySQL Enterprise Backup 3.8 can be restored using MySQL Enterprise Backup 3.7 or 3.6 using the normal restore procedure. To restore a MySQL Enterprise Backup 3.8 backup using MySQL Enterprise Backup 3.5, copy the files and directories from the `datadir` subdirectory of the backup into the main backup directory. MySQL Enterprise Backup 3.5 expects the files to restore to be at the top level of the backup directory.

B.4 Expanded Use of Configuration Files

In MySQL Enterprise Backup 3.5 and earlier, only a limited set of configuration parameters were recognized in the `my.cnf` file, and the backup commands required the paths to one or two configuration files as command-line parameters. In MySQL Enterprise Backup 3.6 and higher, many more parameters are recognized from the configuration file, and the configuration file is automatically located using the same mechanism as the `mysqld` server. For example, connection settings for the database can now be read from the configuration file rather than specified as command-line parameters. Settings such as `innodb_data_home_dir` are now determined from the database connection, rather than required to be specified in the configuration file. Because of the enhanced processing of configuration files and additional command-line options, the second configuration file used by the former `ibbackup` command is no longer needed.

B.5 Relative and Absolute Paths

Prior to MySQL Enterprise Backup 3.6, all file specifications for backup and restore used absolute paths. In MySQL Enterprise Backup 3.6 and higher, you can specify a top-level directory for backups, and the backup process constructs relative paths underneath that directory.

B.6 New and Changed Options in MySQL Enterprise Backup 3.6 and Higher

Table B.1 New and Changed `mysqlbackup` Options in MySQL Enterprise Backup 3.6 and Higher

Old Option	New Option	Notes
`--lsn=LSN`	`--start-lsn=LSN`	The option name is changed for clarity.
`--use-memory=MB`	`--limit-memory=MB`	The option name is changed for clarity.
`--compress[=LEVEL]`	`--compress` and `--compress-level=LEVEL`	The former single option is split into two, with an explicit option to enable compression.
`--no-timestamp`	`--with-timestamp`	The default is reversed: no timestamp subdirectory is created. To preserve the former behavior, specify `--with-timestamp` to put the backup data in a subdirectory named based on the backup timestamp.
	`--backup-dir=PATH`	New option in MySQL Enterprise Backup 3.6.
	`--backup-image=IMAGE`	New option in MySQL Enterprise Backup 3.6.
	`--only-innodb`	New option in MySQL Enterprise Backup 3.6.
	`--only-innodb-with-frm`	New option in MySQL Enterprise Backup 3.7.
	`--no-history-logging`	New option in MySQL Enterprise Backup 3.6.
	`--no-connection`	New option in MySQL Enterprise Backup 3.6.
	`--connect-if-online`	New option in MySQL Enterprise Backup 3.6.
	`--no-locking`	New option in MySQL Enterprise Backup 3.6.
	`--databases-list-file=LIST`	New option in MySQL Enterprise Backup 3.6.
	`--comments`	New option in MySQL Enterprise Backup 3.6.
	`--comments-file=PATH`	New option in MySQL Enterprise Backup 3.6.
`--copy-back`	`copy-back`	This option is promoted to a mode of operation. Instead of specifying a configuration file and a path to the backup data, now you specify the location of the backup data with the `--backup-dir` option, and the configuration parameters

Old Option	New Option	Notes
		are read from your default configuration file.
`--apply-log`	`apply-log`	This option is promoted to a mode of operation. Instead of specifying a configuration file, now you specify the location of the backup data with the `--backup-dir` option, and the configuration parameters are read from the `backup-my.cnf` configuration file that `mysqlbackup` creates in that directory.
`--apply-log --incremental`	`apply-incremental-backup`	This former combination of options is promoted to a mode of operation. Instead of supplying the paths of two user-created configuration files, you specify the location of the incremental and full backup directories with the `--incremental-backup-dir` and `--backup-dir` options. The configuration parameters are read from the `backup-my.cnf` files that `mysqlbackup` creates in those directories.
	`--incremental-base`	New option in MySQL Enterprise Backup 3.7.
	`--incremental-with-redo-log-only`	New option in MySQL Enterprise Backup 3.7.
	`validate`	New option in MySQL Enterprise Backup 3.7.
	`--sbt-environment`	New option in MySQL Enterprise Backup 3.7.

B.7 Comparison of MySQL Enterprise Backup and InnoDB Hot Backup

In terms of features, the MySQL Enterprise Backup product is a superset of the InnoDB Hot Backup product that it supersedes:

- The `mysqlbackup` command, a cross-platform replacement for the `innobackup` command, is now available on Windows. Windows users can back up tables from other storage engines besides InnoDB, such as MyISAM tables, without writing their own wrapper scripts.
- The `mysqlbackup` command now includes all the capabilities of the former `ibbackup` command, making that command obsolete.

  This documentation refers to the `mysqlbackup` command exclusively.
- The `mysqlbackup` command is a C program connecting to the server through the MySQL API, rather than a Perl script that runs the `mysql` command. Because it does not run the actual `mysql`

command, it does not support the `--mysql-extra-args` option of the `innobackup`, but otherwise the syntax is compatible.

If this implementation change presents any issues for former users of the InnoDB Hot Backup product (for example, if you customized the `innobackup` script or relied on specific `mysqld` options passed through the `--mysql-extra-args` option), please submit requirements against the new `mysqlbackup` command.

- Currently, the old `ibbackup` and `innobackup` commands are still supplied as aliases or copies of the `mysqlbackup` command. When `mysqlbackup` is run under these names, it accepts the same old option syntax from those commands. This backward compatibility is for troubleshooting in case of upgrade issues as you transition to the `mysqlbackup` command.

  > **Note**
  >
  > Starting in MySQL Enterprise Backup 3.7.1, the `innobackup` and `ibbackup` commands that are aliases of the `mysqlbackup` display warning messages when you use them. These aliased commands are expected to be removed in a future release.

- Backups produced by the InnoDB Hot Backup product can be restored by the MySQL Enterprise Backup product.
- The streaming backup feature is new to MySQL Enterprise Backup.
- The single-file backup feature is new to MySQL Enterprise Backup.
- The incremental backup feature is new to MySQL Enterprise Backup.
- Support for the Barracuda file format is new to MySQL Enterprise Backup. Once you upgrade your database servers to MySQL 5.1 with the InnoDB Plugin, or MySQL 5.5 and higher where support for the new file format is built in, you need to use MySQL Enterprise Backup to ensure you can back up all InnoDB tables.
- The MySQL Enterprise Backup product includes some new performance optimizations, such as the `posix_fadvise()` system call.
- A new logging capability records the progress of running backup jobs, and historical details for completed backup jobs. For details, see Section 9.4, “Using the MySQL Enterprise Backup Logs”.
- The `mysqlbackup` command has extra flexibility for specifying the MySQL connection information. It can read the user, password, port and socket options from the `[client]` group of your default or user-specified configuration file. If you supply the `--password` option without an argument, you are prompted to enter the password interactively.
- The optimization within the `ibbackup` command that skipped copying unused space within InnoDB tablespace files, is available within `mysqlbackup` only in combination with the `--compressed` option. Use compressed backups if this storage overhead is significant for your data.

B.8 `ibbackup` and `innobackup` Commands

For convenience while upgrading to the latest `mysqlbackup` syntax, you can use the previous `ibbackup` and `innobackup` command names and syntax. When the `mysqlbackup` command is run under one of those other names, either through a symbolic link or by copying the executable file to a new filename, it supports the same option syntax, output filenames, and other behavior as in MySQL Enterprise Backup 3.5. These alternative command names are included in the MySQL Enterprise Backup installation, using the appropriate mechanism for each operating system. For information about the older command names and option syntax, see MySQL Enterprise Backup User's Guide (Version 3.5.4).

> **Important**
>
> We strongly advise our customers to upgrade to the new `mysqlbackup` syntax. We intend to deprecate the old `ibbackup` and `innobackup` syntax soon.

Example B.1 Simple Backup Emulating `ibbackup` Behavior

If you have older scripts that use the `ibbackup` command with 2 configuration files specified on the command line, a corresponding `mysqlbackup` command looks like:

```
mysqlbackup --only-innodb --no-connection --backup-dir=/path/to/backup backup
```

The above command does not back up `.frm` and MyISAM files as `ibbackup` does.

The `my.cnf` must include 6 essential parameters in the `[mysqld]` section or in the `[mysqlbackup]` section. For example, the `my.cnf` might look like:

```
[mysqld]
datadir = /backup/mysql
innodb_data_file_path = ibdata1:256M;ibdata2:256M:autoextend
innodb_log_group_home_dir = /backup/mysql/innodb/log
innodb_data_home_dir = /backup/mysql/innodb/data
innodb_log_file_size = 256M
innodb_log_files_in_group = 3

[mysqlbackup]
backup_innodb_log_group_home_dir = /backup/mysql/innodb/log         [Optional]
backup_innodb_data_file_path = ibdata1:256M;ibdata2:256M:autoextend [Optional]
backup_innodb_data_home_dir = /backup/mysql/innodb/data             [Optional]
backup_innodb_log_file_size = 256M                                  [Optional]
backup_innodb_log_files_in_group = 3                                [Optional]
```

The `backup_innodb_*` options typically have the same values as the corresponding `innodb_*` options, in which case you do not need to specify them.

B.9 Cross-Platform Compatibility

MySQL Enterprise Backup is cross-platform compatible when running on the Linux and Windows operating systems: backups on a Linux machine can be restored on a Windows machine, and vice versa. However, to avoid data transfer problems arising from letter cases of database or table names, the variable `lower_case_table_names` must be properly configured on the MySQL servers. For details, see Identifier Case Sensitivity.

Appendix C Extended Examples

Table of Contents

This section illustrates the commands and associated output for various backup and restore operations.

C.1 Sample Directory Structure for Full Backup

Here is an example of the subdirectories and files underneath a typical backup directory. The `--with-timestamp` option creates a new subdirectory for each backup, named according to the timestamp of the job. This example shows a backup of the databases from an installation of the MySQL Enterprise Monitor product, which like MySQL Enterprise Backup is available to customers with MySQL Enterprise subscriptions. The backups contain the files for the InnoDB system tablespace, `.idb`, `.frm`, `.MYD`, `.MYI`, `.CSV`, and `.CSM` files representing table and index data from various storage engines, and `.par` and `#P#` files representing partitioned tables.

```
$ find ~/backups
/Users/cirrus/backups
/Users/cirrus/backups/2011-06-16_10-33-47
/Users/cirrus/backups/2011-06-16_10-33-47/backup-my.cnf
/Users/cirrus/backups/2011-06-16_10-33-47/datadir
/Users/cirrus/backups/2011-06-16_10-33-47/datadir/ib_logfile0
/Users/cirrus/backups/2011-06-16_10-33-47/datadir/ib_logfile1
/Users/cirrus/backups/2011-06-16_10-33-47/datadir/ibbackup_logfile
/Users/cirrus/backups/2011-06-16_10-33-47/datadir/ibdata1
/Users/cirrus/backups/2011-06-16_10-33-47/datadir/mem
/Users/cirrus/backups/2011-06-16_10-33-47/datadir/mem/db.opt
/Users/cirrus/backups/2011-06-16_10-33-47/datadir/mem/dc_p_double#P#p0.ibd
/Users/cirrus/backups/2011-06-16_10-33-47/datadir/mem/dc_p_double#P#p1.ibd
/Users/cirrus/backups/2011-06-16_10-33-47/datadir/mem/dc_p_double#P#p2.ibd
/Users/cirrus/backups/2011-06-16_10-33-47/datadir/mem/dc_p_double#P#p3.ibd
/Users/cirrus/backups/2011-06-16_10-33-47/datadir/mem/dc_p_double#P#p4.ibd
/Users/cirrus/backups/2011-06-16_10-33-47/datadir/mem/dc_p_double#P#p5.ibd
/Users/cirrus/backups/2011-06-16_10-33-47/datadir/mem/dc_p_double#P#p6.ibd
/Users/cirrus/backups/2011-06-16_10-33-47/datadir/mem/dc_p_double#P#p7.ibd
/Users/cirrus/backups/2011-06-16_10-33-47/datadir/mem/dc_p_double#P#p8.ibd
/Users/cirrus/backups/2011-06-16_10-33-47/datadir/mem/dc_p_double.frm
/Users/cirrus/backups/2011-06-16_10-33-47/datadir/mem/dc_p_double.par
/Users/cirrus/backups/2011-06-16_10-33-47/datadir/mem/dc_p_long#P#p0.ibd
/Users/cirrus/backups/2011-06-16_10-33-47/datadir/mem/dc_p_long#P#p1.ibd
/Users/cirrus/backups/2011-06-16_10-33-47/datadir/mem/dc_p_long#P#p2.ibd
/Users/cirrus/backups/2011-06-16_10-33-47/datadir/mem/dc_p_long#P#p3.ibd
/Users/cirrus/backups/2011-06-16_10-33-47/datadir/mem/dc_p_long#P#p4.ibd
/Users/cirrus/backups/2011-06-16_10-33-47/datadir/mem/dc_p_long#P#p5.ibd
/Users/cirrus/backups/2011-06-16_10-33-47/datadir/mem/dc_p_long#P#p6.ibd
/Users/cirrus/backups/2011-06-16_10-33-47/datadir/mem/dc_p_long#P#p7.ibd
/Users/cirrus/backups/2011-06-16_10-33-47/datadir/mem/dc_p_long#P#p8.ibd
/Users/cirrus/backups/2011-06-16_10-33-47/datadir/mem/dc_p_long.frm
/Users/cirrus/backups/2011-06-16_10-33-47/datadir/mem/dc_p_long.par
/Users/cirrus/backups/2011-06-16_10-33-47/datadir/mem/dc_p_string#P#p0.ibd
/Users/cirrus/backups/2011-06-16_10-33-47/datadir/mem/dc_p_string#P#p1.ibd
/Users/cirrus/backups/2011-06-16_10-33-47/datadir/mem/dc_p_string#P#p2.ibd
/Users/cirrus/backups/2011-06-16_10-33-47/datadir/mem/dc_p_string#P#p3.ibd
/Users/cirrus/backups/2011-06-16_10-33-47/datadir/mem/dc_p_string#P#p4.ibd
/Users/cirrus/backups/2011-06-16_10-33-47/datadir/mem/dc_p_string#P#p5.ibd
/Users/cirrus/backups/2011-06-16_10-33-47/datadir/mem/dc_p_string#P#p6.ibd
/Users/cirrus/backups/2011-06-16_10-33-47/datadir/mem/dc_p_string#P#p7.ibd
/Users/cirrus/backups/2011-06-16_10-33-47/datadir/mem/dc_p_string#P#p8.ibd
/Users/cirrus/backups/2011-06-16_10-33-47/datadir/mem/dc_p_string.frm
```

```
/Users/cirrus/backups/2011-06-16_10-33-47/datadir/mem/dc_p_string.par
/Users/cirrus/backups/2011-06-16_10-33-47/datadir/mem/graph_dc_schedules.frm
/Users/cirrus/backups/2011-06-16_10-33-47/datadir/mem/graph_dc_schedules.ibd
/Users/cirrus/backups/2011-06-16_10-33-47/datadir/mem/graph_schedules.frm
/Users/cirrus/backups/2011-06-16_10-33-47/datadir/mem/graph_schedules.ibd
/Users/cirrus/backups/2011-06-16_10-33-47/datadir/mem/graph_series_v2.frm
/Users/cirrus/backups/2011-06-16_10-33-47/datadir/mem/graph_series_v2.ibd
/Users/cirrus/backups/2011-06-16_10-33-47/datadir/mem/graph_tags.frm
/Users/cirrus/backups/2011-06-16_10-33-47/datadir/mem/graph_tags.ibd
/Users/cirrus/backups/2011-06-16_10-33-47/datadir/mem/graph_variables_v2.frm
/Users/cirrus/backups/2011-06-16_10-33-47/datadir/mem/graph_variables_v2.ibd
/Users/cirrus/backups/2011-06-16_10-33-47/datadir/mem/graphs.frm
/Users/cirrus/backups/2011-06-16_10-33-47/datadir/mem/graphs.ibd
/Users/cirrus/backups/2011-06-16_10-33-47/datadir/mem/group_members_v2.frm
/Users/cirrus/backups/2011-06-16_10-33-47/datadir/mem/group_members_v2.ibd
/Users/cirrus/backups/2011-06-16_10-33-47/datadir/mem/group_names.frm
/Users/cirrus/backups/2011-06-16_10-33-47/datadir/mem/group_names.ibd
/Users/cirrus/backups/2011-06-16_10-33-47/datadir/mem/hilo_sequence_iia.frm
/Users/cirrus/backups/2011-06-16_10-33-47/datadir/mem/hilo_sequence_iia.ibd
/Users/cirrus/backups/2011-06-16_10-33-47/datadir/mem/hilo_sequence_inventory_attributes.frm
/Users/cirrus/backups/2011-06-16_10-33-47/datadir/mem/hilo_sequence_inventory_attributes.ibd
/Users/cirrus/backups/2011-06-16_10-33-47/datadir/mem/hilo_sequence_inventory_instances.frm
/Users/cirrus/backups/2011-06-16_10-33-47/datadir/mem/hilo_sequence_inventory_instances.ibd
/Users/cirrus/backups/2011-06-16_10-33-47/datadir/mem/hilo_sequence_inventory_namespaces.frm
/Users/cirrus/backups/2011-06-16_10-33-47/datadir/mem/hilo_sequence_inventory_namespaces.ibd
/Users/cirrus/backups/2011-06-16_10-33-47/datadir/mem/hilo_sequence_inventory_types.frm
/Users/cirrus/backups/2011-06-16_10-33-47/datadir/mem/hilo_sequence_inventory_types.ibd
/Users/cirrus/backups/2011-06-16_10-33-47/datadir/mem/hilo_sequence_rule_alarms.frm
/Users/cirrus/backups/2011-06-16_10-33-47/datadir/mem/hilo_sequence_rule_alarms.ibd
/Users/cirrus/backups/2011-06-16_10-33-47/datadir/mem/hilo_sequence_rule_eval_results.frm
/Users/cirrus/backups/2011-06-16_10-33-47/datadir/mem/hilo_sequence_rule_eval_results.ibd
/Users/cirrus/backups/2011-06-16_10-33-47/datadir/mem/inventory_attributes.frm
/Users/cirrus/backups/2011-06-16_10-33-47/datadir/mem/inventory_attributes.ibd
/Users/cirrus/backups/2011-06-16_10-33-47/datadir/mem/inventory_instance_attributes.frm
/Users/cirrus/backups/2011-06-16_10-33-47/datadir/mem/inventory_instance_attributes.ibd
/Users/cirrus/backups/2011-06-16_10-33-47/datadir/mem/inventory_instance_tags.frm
/Users/cirrus/backups/2011-06-16_10-33-47/datadir/mem/inventory_instance_tags.ibd
/Users/cirrus/backups/2011-06-16_10-33-47/datadir/mem/inventory_instances.frm
/Users/cirrus/backups/2011-06-16_10-33-47/datadir/mem/inventory_instances.ibd
/Users/cirrus/backups/2011-06-16_10-33-47/datadir/mem/inventory_namespaces.frm
/Users/cirrus/backups/2011-06-16_10-33-47/datadir/mem/inventory_namespaces.ibd
/Users/cirrus/backups/2011-06-16_10-33-47/datadir/mem/inventory_types.frm
/Users/cirrus/backups/2011-06-16_10-33-47/datadir/mem/inventory_types.ibd
/Users/cirrus/backups/2011-06-16_10-33-47/datadir/mem/log_db_actions#P#p0.ibd
/Users/cirrus/backups/2011-06-16_10-33-47/datadir/mem/log_db_actions#P#p1.ibd
/Users/cirrus/backups/2011-06-16_10-33-47/datadir/mem/log_db_actions#P#p2.ibd
/Users/cirrus/backups/2011-06-16_10-33-47/datadir/mem/log_db_actions#P#p3.ibd
/Users/cirrus/backups/2011-06-16_10-33-47/datadir/mem/log_db_actions#P#p4.ibd
/Users/cirrus/backups/2011-06-16_10-33-47/datadir/mem/log_db_actions#P#p5.ibd
/Users/cirrus/backups/2011-06-16_10-33-47/datadir/mem/log_db_actions#P#p6.ibd
/Users/cirrus/backups/2011-06-16_10-33-47/datadir/mem/log_db_actions#P#p7.ibd
/Users/cirrus/backups/2011-06-16_10-33-47/datadir/mem/log_db_actions#P#p8.ibd
/Users/cirrus/backups/2011-06-16_10-33-47/datadir/mem/log_db_actions.frm
/Users/cirrus/backups/2011-06-16_10-33-47/datadir/mem/log_db_actions.par
/Users/cirrus/backups/2011-06-16_10-33-47/datadir/mem/loghistogram_data.frm
/Users/cirrus/backups/2011-06-16_10-33-47/datadir/mem/loghistogram_data.ibd
/Users/cirrus/backups/2011-06-16_10-33-47/datadir/mem/map_entries.frm
/Users/cirrus/backups/2011-06-16_10-33-47/datadir/mem/map_entries.ibd
/Users/cirrus/backups/2011-06-16_10-33-47/datadir/mem/migration.frm
/Users/cirrus/backups/2011-06-16_10-33-47/datadir/mem/migration.ibd
/Users/cirrus/backups/2011-06-16_10-33-47/datadir/mem/migration_migration_state.frm
/Users/cirrus/backups/2011-06-16_10-33-47/datadir/mem/migration_migration_state.ibd
/Users/cirrus/backups/2011-06-16_10-33-47/datadir/mem/migration_migration_status_servers.frm
/Users/cirrus/backups/2011-06-16_10-33-47/datadir/mem/migration_migration_status_servers.ibd
/Users/cirrus/backups/2011-06-16_10-33-47/datadir/mem/migration_state.frm
/Users/cirrus/backups/2011-06-16_10-33-47/datadir/mem/migration_state.ibd
/Users/cirrus/backups/2011-06-16_10-33-47/datadir/mem/migration_status_data_collection.frm
/Users/cirrus/backups/2011-06-16_10-33-47/datadir/mem/migration_status_data_collection.ibd
/Users/cirrus/backups/2011-06-16_10-33-47/datadir/mem/migration_status_servers.frm
/Users/cirrus/backups/2011-06-16_10-33-47/datadir/mem/migration_status_servers.ibd
/Users/cirrus/backups/2011-06-16_10-33-47/datadir/mem/migration_status_servers_migration_state.frm
/Users/cirrus/backups/2011-06-16_10-33-47/datadir/mem/migration_status_servers_migration_state.ibd
```

```
/Users/cirrus/backups/2011-06-16_10-33-47/datadir/mem/migration_status_servers_migration_status_data_co
/Users/cirrus/backups/2011-06-16_10-33-47/datadir/mem/migration_status_servers_migration_status_data_co
/Users/cirrus/backups/2011-06-16_10-33-47/datadir/mem/mos_service_requests.frm
/Users/cirrus/backups/2011-06-16_10-33-47/datadir/mem/mos_service_requests.ibd
/Users/cirrus/backups/2011-06-16_10-33-47/datadir/mem/resource_bundle.frm
/Users/cirrus/backups/2011-06-16_10-33-47/datadir/mem/resource_bundle.ibd
/Users/cirrus/backups/2011-06-16_10-33-47/datadir/mem/resource_bundle_map.frm
/Users/cirrus/backups/2011-06-16_10-33-47/datadir/mem/resource_bundle_map.ibd
/Users/cirrus/backups/2011-06-16_10-33-47/datadir/mem/rule_alarms.frm
/Users/cirrus/backups/2011-06-16_10-33-47/datadir/mem/rule_alarms.ibd
/Users/cirrus/backups/2011-06-16_10-33-47/datadir/mem/rule_dc_schedules.frm
/Users/cirrus/backups/2011-06-16_10-33-47/datadir/mem/rule_dc_schedules.ibd
/Users/cirrus/backups/2011-06-16_10-33-47/datadir/mem/rule_eval_result_vars.frm
/Users/cirrus/backups/2011-06-16_10-33-47/datadir/mem/rule_eval_result_vars.ibd
/Users/cirrus/backups/2011-06-16_10-33-47/datadir/mem/rule_eval_results.frm
/Users/cirrus/backups/2011-06-16_10-33-47/datadir/mem/rule_eval_results.ibd
/Users/cirrus/backups/2011-06-16_10-33-47/datadir/mem/rule_schedule_email_targets.frm
/Users/cirrus/backups/2011-06-16_10-33-47/datadir/mem/rule_schedule_email_targets.ibd
/Users/cirrus/backups/2011-06-16_10-33-47/datadir/mem/rule_schedules.frm
/Users/cirrus/backups/2011-06-16_10-33-47/datadir/mem/rule_schedules.ibd
/Users/cirrus/backups/2011-06-16_10-33-47/datadir/mem/rule_tags.frm
/Users/cirrus/backups/2011-06-16_10-33-47/datadir/mem/rule_tags.ibd
/Users/cirrus/backups/2011-06-16_10-33-47/datadir/mem/rule_thresholds.frm
/Users/cirrus/backups/2011-06-16_10-33-47/datadir/mem/rule_thresholds.ibd
/Users/cirrus/backups/2011-06-16_10-33-47/datadir/mem/rule_variables.frm
/Users/cirrus/backups/2011-06-16_10-33-47/datadir/mem/rule_variables.ibd
/Users/cirrus/backups/2011-06-16_10-33-47/datadir/mem/rules.frm
/Users/cirrus/backups/2011-06-16_10-33-47/datadir/mem/rules.ibd
/Users/cirrus/backups/2011-06-16_10-33-47/datadir/mem/schema_version_v2.frm
/Users/cirrus/backups/2011-06-16_10-33-47/datadir/mem/schema_version_v2.ibd
/Users/cirrus/backups/2011-06-16_10-33-47/datadir/mem/statement_data.frm
/Users/cirrus/backups/2011-06-16_10-33-47/datadir/mem/statement_data.ibd
/Users/cirrus/backups/2011-06-16_10-33-47/datadir/mem/statement_examples.frm
/Users/cirrus/backups/2011-06-16_10-33-47/datadir/mem/statement_examples.ibd
/Users/cirrus/backups/2011-06-16_10-33-47/datadir/mem/statement_explain_data.frm
/Users/cirrus/backups/2011-06-16_10-33-47/datadir/mem/statement_explain_data.ibd
/Users/cirrus/backups/2011-06-16_10-33-47/datadir/mem/statement_summaries.frm
/Users/cirrus/backups/2011-06-16_10-33-47/datadir/mem/statement_summaries.ibd
/Users/cirrus/backups/2011-06-16_10-33-47/datadir/mem/statement_summary_data.frm
/Users/cirrus/backups/2011-06-16_10-33-47/datadir/mem/statement_summary_data.ibd
/Users/cirrus/backups/2011-06-16_10-33-47/datadir/mem/system_maps.frm
/Users/cirrus/backups/2011-06-16_10-33-47/datadir/mem/system_maps.ibd
/Users/cirrus/backups/2011-06-16_10-33-47/datadir/mem/tags.frm
/Users/cirrus/backups/2011-06-16_10-33-47/datadir/mem/tags.ibd
/Users/cirrus/backups/2011-06-16_10-33-47/datadir/mem/target_email.frm
/Users/cirrus/backups/2011-06-16_10-33-47/datadir/mem/target_email.ibd
/Users/cirrus/backups/2011-06-16_10-33-47/datadir/mem/user_form_defaults.frm
/Users/cirrus/backups/2011-06-16_10-33-47/datadir/mem/user_form_defaults.ibd
/Users/cirrus/backups/2011-06-16_10-33-47/datadir/mem/user_preferences.frm
/Users/cirrus/backups/2011-06-16_10-33-47/datadir/mem/user_preferences.ibd
/Users/cirrus/backups/2011-06-16_10-33-47/datadir/mem/user_tags.frm
/Users/cirrus/backups/2011-06-16_10-33-47/datadir/mem/user_tags.ibd
/Users/cirrus/backups/2011-06-16_10-33-47/datadir/mem/users.frm
/Users/cirrus/backups/2011-06-16_10-33-47/datadir/mem/users.ibd
/Users/cirrus/backups/2011-06-16_10-33-47/datadir/mem/whats_new_entries.frm
/Users/cirrus/backups/2011-06-16_10-33-47/datadir/mem/whats_new_entries.ibd
/Users/cirrus/backups/2011-06-16_10-33-47/datadir/mysql
/Users/cirrus/backups/2011-06-16_10-33-47/datadir/mysql/backup_history.CSM
/Users/cirrus/backups/2011-06-16_10-33-47/datadir/mysql/backup_history.CSV
/Users/cirrus/backups/2011-06-16_10-33-47/datadir/mysql/backup_history.frm
/Users/cirrus/backups/2011-06-16_10-33-47/datadir/mysql/backup_progress.CSM
/Users/cirrus/backups/2011-06-16_10-33-47/datadir/mysql/backup_progress.CSV
/Users/cirrus/backups/2011-06-16_10-33-47/datadir/mysql/backup_progress.frm
/Users/cirrus/backups/2011-06-16_10-33-47/datadir/mysql/columns_priv.frm
/Users/cirrus/backups/2011-06-16_10-33-47/datadir/mysql/columns_priv.MYD
/Users/cirrus/backups/2011-06-16_10-33-47/datadir/mysql/columns_priv.MYI
/Users/cirrus/backups/2011-06-16_10-33-47/datadir/mysql/db.frm
/Users/cirrus/backups/2011-06-16_10-33-47/datadir/mysql/db.MYD
/Users/cirrus/backups/2011-06-16_10-33-47/datadir/mysql/db.MYI
/Users/cirrus/backups/2011-06-16_10-33-47/datadir/mysql/event.frm
/Users/cirrus/backups/2011-06-16_10-33-47/datadir/mysql/event.MYD
/Users/cirrus/backups/2011-06-16_10-33-47/datadir/mysql/event.MYI
```

```
/Users/cirrus/backups/2011-06-16_10-33-47/datadir/mysql/func.frm
/Users/cirrus/backups/2011-06-16_10-33-47/datadir/mysql/func.MYD
/Users/cirrus/backups/2011-06-16_10-33-47/datadir/mysql/func.MYI
/Users/cirrus/backups/2011-06-16_10-33-47/datadir/mysql/general_log.CSM
/Users/cirrus/backups/2011-06-16_10-33-47/datadir/mysql/general_log.CSV
/Users/cirrus/backups/2011-06-16_10-33-47/datadir/mysql/general_log.frm
/Users/cirrus/backups/2011-06-16_10-33-47/datadir/mysql/help_category.frm
/Users/cirrus/backups/2011-06-16_10-33-47/datadir/mysql/help_category.MYD
/Users/cirrus/backups/2011-06-16_10-33-47/datadir/mysql/help_category.MYI
/Users/cirrus/backups/2011-06-16_10-33-47/datadir/mysql/help_keyword.frm
/Users/cirrus/backups/2011-06-16_10-33-47/datadir/mysql/help_keyword.MYD
/Users/cirrus/backups/2011-06-16_10-33-47/datadir/mysql/help_keyword.MYI
/Users/cirrus/backups/2011-06-16_10-33-47/datadir/mysql/help_relation.frm
/Users/cirrus/backups/2011-06-16_10-33-47/datadir/mysql/help_relation.MYD
/Users/cirrus/backups/2011-06-16_10-33-47/datadir/mysql/help_relation.MYI
/Users/cirrus/backups/2011-06-16_10-33-47/datadir/mysql/help_topic.frm
/Users/cirrus/backups/2011-06-16_10-33-47/datadir/mysql/help_topic.MYD
/Users/cirrus/backups/2011-06-16_10-33-47/datadir/mysql/help_topic.MYI
/Users/cirrus/backups/2011-06-16_10-33-47/datadir/mysql/host.frm
/Users/cirrus/backups/2011-06-16_10-33-47/datadir/mysql/host.MYD
/Users/cirrus/backups/2011-06-16_10-33-47/datadir/mysql/host.MYI
/Users/cirrus/backups/2011-06-16_10-33-47/datadir/mysql/ibbackup_binlog_marker.frm
/Users/cirrus/backups/2011-06-16_10-33-47/datadir/mysql/ibbackup_binlog_marker.ibd
/Users/cirrus/backups/2011-06-16_10-33-47/datadir/mysql/inventory.frm
/Users/cirrus/backups/2011-06-16_10-33-47/datadir/mysql/inventory.MYD
/Users/cirrus/backups/2011-06-16_10-33-47/datadir/mysql/inventory.MYI
/Users/cirrus/backups/2011-06-16_10-33-47/datadir/mysql/ndb_binlog_index.frm
/Users/cirrus/backups/2011-06-16_10-33-47/datadir/mysql/ndb_binlog_index.MYD
/Users/cirrus/backups/2011-06-16_10-33-47/datadir/mysql/ndb_binlog_index.MYI
/Users/cirrus/backups/2011-06-16_10-33-47/datadir/mysql/plugin.frm
/Users/cirrus/backups/2011-06-16_10-33-47/datadir/mysql/plugin.MYD
/Users/cirrus/backups/2011-06-16_10-33-47/datadir/mysql/plugin.MYI
/Users/cirrus/backups/2011-06-16_10-33-47/datadir/mysql/proc.frm
/Users/cirrus/backups/2011-06-16_10-33-47/datadir/mysql/proc.MYD
/Users/cirrus/backups/2011-06-16_10-33-47/datadir/mysql/proc.MYI
/Users/cirrus/backups/2011-06-16_10-33-47/datadir/mysql/procs_priv.frm
/Users/cirrus/backups/2011-06-16_10-33-47/datadir/mysql/procs_priv.MYD
/Users/cirrus/backups/2011-06-16_10-33-47/datadir/mysql/procs_priv.MYI
/Users/cirrus/backups/2011-06-16_10-33-47/datadir/mysql/servers.frm
/Users/cirrus/backups/2011-06-16_10-33-47/datadir/mysql/servers.MYD
/Users/cirrus/backups/2011-06-16_10-33-47/datadir/mysql/servers.MYI
/Users/cirrus/backups/2011-06-16_10-33-47/datadir/mysql/slow_log.CSM
/Users/cirrus/backups/2011-06-16_10-33-47/datadir/mysql/slow_log.CSV
/Users/cirrus/backups/2011-06-16_10-33-47/datadir/mysql/slow_log.frm
/Users/cirrus/backups/2011-06-16_10-33-47/datadir/mysql/tables_priv.frm
/Users/cirrus/backups/2011-06-16_10-33-47/datadir/mysql/tables_priv.MYD
/Users/cirrus/backups/2011-06-16_10-33-47/datadir/mysql/tables_priv.MYI
/Users/cirrus/backups/2011-06-16_10-33-47/datadir/mysql/time_zone.frm
/Users/cirrus/backups/2011-06-16_10-33-47/datadir/mysql/time_zone.MYD
/Users/cirrus/backups/2011-06-16_10-33-47/datadir/mysql/time_zone.MYI
/Users/cirrus/backups/2011-06-16_10-33-47/datadir/mysql/time_zone_leap_second.frm
/Users/cirrus/backups/2011-06-16_10-33-47/datadir/mysql/time_zone_leap_second.MYD
/Users/cirrus/backups/2011-06-16_10-33-47/datadir/mysql/time_zone_leap_second.MYI
/Users/cirrus/backups/2011-06-16_10-33-47/datadir/mysql/time_zone_name.frm
/Users/cirrus/backups/2011-06-16_10-33-47/datadir/mysql/time_zone_name.MYD
/Users/cirrus/backups/2011-06-16_10-33-47/datadir/mysql/time_zone_name.MYI
/Users/cirrus/backups/2011-06-16_10-33-47/datadir/mysql/time_zone_transition.frm
/Users/cirrus/backups/2011-06-16_10-33-47/datadir/mysql/time_zone_transition.MYD
/Users/cirrus/backups/2011-06-16_10-33-47/datadir/mysql/time_zone_transition.MYI
/Users/cirrus/backups/2011-06-16_10-33-47/datadir/mysql/time_zone_transition_type.frm
/Users/cirrus/backups/2011-06-16_10-33-47/datadir/mysql/time_zone_transition_type.MYD
/Users/cirrus/backups/2011-06-16_10-33-47/datadir/mysql/time_zone_transition_type.MYI
/Users/cirrus/backups/2011-06-16_10-33-47/datadir/mysql/user.frm
/Users/cirrus/backups/2011-06-16_10-33-47/datadir/mysql/user.MYD
/Users/cirrus/backups/2011-06-16_10-33-47/datadir/mysql/user.MYI
/Users/cirrus/backups/2011-06-16_10-33-47/meta
/Users/cirrus/backups/2011-06-16_10-33-47/meta/backup_content.xml
/Users/cirrus/backups/2011-06-16_10-33-47/meta/backup_create.xml
/Users/cirrus/backups/2011-06-16_10-33-47/meta/backup_variables.txt
/Users/cirrus/backups/2011-06-16_10-34-12
/Users/cirrus/backups/2011-06-16_10-34-12/backup-my.cnf
/Users/cirrus/backups/2011-06-16_10-34-12/datadir
```

```
/Users/cirrus/backups/2011-06-16_10-34-12/datadir/ib_logfile0
/Users/cirrus/backups/2011-06-16_10-34-12/datadir/ib_logfile1
/Users/cirrus/backups/2011-06-16_10-34-12/datadir/ibbackup_logfile
/Users/cirrus/backups/2011-06-16_10-34-12/datadir/ibdata1
/Users/cirrus/backups/2011-06-16_10-34-12/datadir/mem
/Users/cirrus/backups/2011-06-16_10-34-12/datadir/mem/db.opt
/Users/cirrus/backups/2011-06-16_10-34-12/datadir/mem/dc_p_double#P#p0.ibd
...same database and table files as the previous backup...
/Users/cirrus/backups/2011-06-16_10-34-12/meta
/Users/cirrus/backups/2011-06-16_10-34-12/meta/backup_content.xml
/Users/cirrus/backups/2011-06-16_10-34-12/meta/backup_create.xml
/Users/cirrus/backups/2011-06-16_10-34-12/meta/backup_variables.txt
```

C.2 Sample Directory Structure for Compressed Backup

Here is an excerpt from the file listing under *`backup-dir`*`/datadir/mem` for a backup from a MySQL Enterprise Monitor repository database. Notice how the `.ibd` files for InnoDB tables are now compressed to `.ibz` files, while other kinds of files are left unchanged.

```
inventory_types.frm
inventory_types.ibz
log_db_actions#P#p0.ibz
log_db_actions#P#p1.ibz
log_db_actions#P#p2.ibz
log_db_actions#P#p3.ibz
log_db_actions#P#p4.ibz
log_db_actions#P#p5.ibz
log_db_actions#P#p6.ibz
log_db_actions#P#p7.ibz
log_db_actions#P#p8.ibz
log_db_actions.frm
log_db_actions.par
loghistogram_data.frm
loghistogram_data.ibz
```

C.3 Sample Directory Structure for Incremental Backup

An incremental backup produces a directory structure containing a subset of the files from a full backup. All non-InnoDB files such as `*.frm` `*.MYD`, and so on are included. `*.ibd` files are included only if they changed since the full backup, that is, if their maximum logical sequence number is higher than the value specified by the `--start-lsn` option.

```
$ find /tmp/backups
/tmp/backups
/tmp/backups/backup-my.cnf
/tmp/backups/datadir
/tmp/backups/datadir/ibbackup_ibd_files
/tmp/backups/datadir/ibbackup_logfile
/tmp/backups/datadir/ibdata1
/tmp/backups/datadir/mem
/tmp/backups/datadir/mem/db.opt
/tmp/backups/datadir/mem/dc_p_double.frm
/tmp/backups/datadir/mem/dc_p_double.par
/tmp/backups/datadir/mem/dc_p_long.frm
/tmp/backups/datadir/mem/dc_p_long.par
/tmp/backups/datadir/mem/dc_p_string.frm
/tmp/backups/datadir/mem/dc_p_string.par
/tmp/backups/datadir/mem/graph_dc_schedules.frm
/tmp/backups/datadir/mem/graph_schedules.frm
... many more files...
```

C.4 Validating a Single-File Backup Image

This script and associated output shows what you see when using the `validate` option to check that a single-file backup was not corrupted as it was transferred from place to place.

validate Script

```
#! /bin/ksh

# Demonstrate and test the 'validate' subcommand of mysqlbackup,
# which works in combination with single-file backup.

bdir=$(pwd)/backup
full=$(pwd)/backup/single
image=backup.image
block=1234 # Arbitrary block to be corrupted

db564=~/sandboxes/msb_5_6_4
config=$db564/my.sandbox.cnf

mkdir -p $bdir
rm -rf $full
mkdir -p $full

# --backup-dir needed here for temporary work files while producing the single-file backup image.
# --backup-image could be a basic file name or the full path, it'll go under the same directory either way.
mysqlbackup --defaults-file=$config --backup-dir=$full --backup-image=$full/$image backup-to-image

echo; echo "*** Image File for Single-File Backup ***"; echo

ls -l $full

echo; echo "*** We expect this validation operation to succeed ***"; echo

# For validating, we need the fully-qualified filename, --backup-dir doesn't help.
mysqlbackup --backup-image=$full/$image validate

echo; echo "*** Producing corrupted copy of single-image file ***"; echo

dd of=$full/$image bs=8192 conv=notrunc count=1 oseek=$block if=/dev/random

echo; echo "*** We expect this validation operation to fail ***"; echo

mysqlbackup --backup-image=$full/$image validate
```

Output

```
$ validate
MySQL Enterprise Backup version 3.7.0 [2011/12/07]
Copyright (c) 2003, 2011, Oracle and/or its affiliates. All Rights Reserved.

INFO: Starting with following command line ...
 mysqlbackup
        --defaults-file=/Users/cirrus/sandboxes/msb_5_6_4/my.sandbox.cnf
        --backup-dir=/Users/cirrus/downloads/meb37/test/backup/single
        --backup-image=/Users/cirrus/downloads/meb37/test/backup/single/backup.image
        backup-to-image

INFO: Got some server configuration information from running server.

IMPORTANT: Please check that mysqlbackup run completes successfully.
            At the end of a successful 'backup-to-image' run mysqlbackup
            prints "mysqlbackup completed OK!".

--------------------------------------------------------------------
                      Server Repository Options:
--------------------------------------------------------------------
  datadir                         =  /Users/cirrus/sandboxes/msb_5_6_4/data/
  innodb_data_home_dir            =
  innodb_data_file_path           =  ibdata1:10M:autoextend
  innodb_log_group_home_dir       =  /Users/cirrus/sandboxes/msb_5_6_4/data/
  innodb_log_files_in_group       =  2
  innodb_log_file_size            =  5242880
```

```
-----------------------------------------------------------------
                     Backup Config Options:
-----------------------------------------------------------------
  datadir                          =  /Users/cirrus/downloads/meb37/test/backup/single/datadir
  innodb_data_home_dir             =  /Users/cirrus/downloads/meb37/test/backup/single/datadir
  innodb_data_file_path            =  ibdata1:10M:autoextend
  innodb_log_group_home_dir        =  /Users/cirrus/downloads/meb37/test/backup/single/datadir
  innodb_log_files_in_group        =  2
  innodb_log_file_size             =  5242880

Backup Image Path= /Users/cirrus/downloads/meb37/test/backup/single/backup.image
 mysqlbackup: INFO: Unique generated backup id for this is 13234786599252660
 mysqlbackup: INFO: System tablespace file format is Antelope.
 mysqlbackup: INFO: Found checkpoint at lsn 183521501.
 mysqlbackup: INFO: Starting log scan from lsn 183521280.
111209 16:57:39 mysqlbackup: INFO: Copying log...
111209 16:57:40 mysqlbackup: INFO: Log copied, lsn 183521501.
          We wait 1 second before starting copying the data files...
111209 16:57:41 mysqlbackup: INFO: Copying /Users/cirrus/sandboxes/msb_5_6_4/data/ibdata1 (Antelope fil
111209 16:57:47 mysqlbackup: INFO: Copying /Users/cirrus/sandboxes/msb_5_6_4/data/test/innodb_busy.ibd
111209 16:57:47 mysqlbackup: INFO: Copying /Users/cirrus/sandboxes/msb_5_6_4/data/test/innodb_quiet.ibd
 mysqlbackup: INFO: Preparing to lock tables: Connected to mysqld server.
111209 16:57:47 mysqlbackup: INFO: Starting to lock all the tables....
111209 16:57:47 mysqlbackup: INFO: All tables are locked and flushed to disk
 mysqlbackup: INFO: Opening backup source directory '/Users/cirrus/sandboxes/msb_5_6_4/data/'
111209 16:57:47 mysqlbackup: INFO: Starting to backup all files in subdirectories of '/Users/cirrus/san
 mysqlbackup: INFO: Backing up the database directory 'mysql'
 mysqlbackup: INFO: Backing up the database directory 'performance_schema'
 mysqlbackup: INFO: Backing up the database directory 'test'
 mysqlbackup: INFO: Copying innodb data and logs during final stage ...
 mysqlbackup: INFO: A copied database page was modified at 183521501.
          (This is the highest lsn found on page)
          Scanned log up to lsn 183524641.
          Was able to parse the log up to lsn 183524641.
          Maximum page number for a log record 467
111209 16:57:47 mysqlbackup: INFO: All tables unlocked
 mysqlbackup: INFO: All MySQL tables were locked for 0.000 seconds
111209 16:57:48 mysqlbackup: INFO: Full backup completed!
 mysqlbackup: WARNING: backup-image already closed
 mysqlbackup: INFO: Backup image created successfully.:
           Image Path: '/Users/cirrus/downloads/meb37/test/backup/single/backup.image'

------------------------------------------------------------
   Parameters Summary
------------------------------------------------------------
   Start LSN                  : 183521280
   End LSN                    : 183524641
------------------------------------------------------------

mysqlbackup completed OK!

*** Image File for Single-File Backup ***

total 237272
-rw-r--r--  1 cirrus  staff        188 Dec  9 16:57 backup-my.cnf
-rw-r--r--  1 cirrus  staff  121478567 Dec  9 16:57 backup.image
drwx------  3 cirrus  staff        102 Dec  9 16:57 datadir
drwx------  6 cirrus  staff        204 Dec  9 16:57 meta

*** We expect this validation operation to succeed ***

MySQL Enterprise Backup version 3.7.0 [2011/12/07]
Copyright (c) 2003, 2011, Oracle and/or its affiliates. All Rights Reserved.

INFO: Starting with following command line ...
 mysqlbackup
        --backup-image=/Users/cirrus/downloads/meb37/test/backup/single/backup.image
        validate

IMPORTANT: Please check that mysqlbackup run completes successfully.
           At the end of a successful 'validate' run mysqlbackup
           prints "mysqlbackup completed OK!".
```

```
 mysqlbackup: INFO: Validating image ... /Users/cirrus/downloads/meb37/test/backup/single/backup.image
 mysqlbackup: INFO: Backup Image MEB version string: 3.7.0 [2011/12/07]
 mysqlbackup: INFO:   Total files as specified in image: 188
 mysqlbackup: INFO:  Backup Image validation successful.
mysqlbackup completed OK!

*** Producing corrupted copy of single-image file ***

1+0 records in
1+0 records out
8192 bytes transferred in 0.000953 secs (8596382 bytes/sec)

*** We expect this validation operation to fail ***

MySQL Enterprise Backup version 3.7.0 [2011/12/07]
Copyright (c) 2003, 2011, Oracle and/or its affiliates. All Rights Reserved.

INFO: Starting with following command line ...
 mysqlbackup
        --backup-image=/Users/cirrus/downloads/meb37/test/backup/single/backup.image
        validate

IMPORTANT: Please check that mysqlbackup run completes successfully.
            At the end of a successful 'validate' run mysqlbackup
            prints "mysqlbackup completed OK!".

 mysqlbackup: INFO: Validating image ... /Users/cirrus/downloads/meb37/test/backup/single/backup.image
 mysqlbackup: INFO: Backup Image MEB version string: 3.7.0 [2011/12/07]
 mysqlbackup: ERROR: Component file checksum mismatch found!.
 File path: datadir/ibdata1.$_append_$.2
 mysqlbackup: ERROR: Backup image validation failed.
```

Appendix D MySQL Enterprise Backup Change History

Table of Contents

This appendix lists the changes to the MySQL Enterprise Backup product, beginning with the most recent release. Each release section covers added or changed functionality, bug fixes, and known issues, if applicable. All bug fixes are referenced by bug number and include a link to the bug database. Bugs are listed in order of resolution. To find a bug quickly, search by bug number.

D.1 Changes in MySQL Enterprise Backup 3.8.2 (2013-06-18)

Functionality Added or Changed

- MySQL Enterprise Backup has a new `--on-disk-full` command line option. `mysqlbackup` could hang when the disk became full, rather than detecting the low space condition. `mysqlbackup` now monitors disk space when running backup commands, and users can now specify the action to take at a disk-full condition with the `--on-disk-full` option. See Section 4.1.11, “Performance / Scalability / Capacity Options” for details. (Bug #13817288, Bug #13804407)

- MySQL Enterprise Backup has a new progress report feature, which periodically outputs short progress indicators on its operations to user-selected destinations (for example, `stdout`, `stderr`, a file, or other choices). This feature is controlled by the new progress report options described in Section 4.1.12, “Progress Report Options”.

Bugs Fixed

- MySQL Server failed to start after a backup was restored if there had been online DDL transactions on partitioned tables during the time of backup. (Bug #16924499)

- When `--innodb-file-per-table=ON`, if a table was renamed and `backup-to-image` was in progress, `apply-log` would fail when being run on the backup. (Bug #16903973)

- `apply-log` failed if `ALTER TABLE ... REORGANIZE PARTITION` was applied to partitioned InnoDB tables during backup. (Bug #16721824, Bug #16903951)

- `apply-incremental-backup` might fail with an assertion error if the InnoDB tables being backed up were created in Barracuda format and with their KEY_BLOCK_SIZE values different from the `innodb_page_size`. This fix ensures that different KEY_BLOCK_SIZE values are handled properly during incremental backup and `apply-incremental-backup` operations.

  > **Note**
  >
  > Perform incremental backups with MySQL Enterprise Backup 3.8.2 instead of any older version to ensure a successful `apply-incremental-backup`.

  (Bug #16423621, Bug #16842291)

- If a table was renamed following a full backup, a subsequent incremental backup could copy the `.frm` file with the new name, but not the associated `.ibd` file with the new name. After a restore, the InnoDB data dictionary could be in an inconsistent state. This issue primarily occurred if the table was not changed between the full backup and the subsequent incremental backup. (Bug #16262690)
- After a full backup, if a table was renamed and modified, `apply-incremental-backup` would crash when run on the backup directory. (Bug #16262609)
- The value of the binary log position in `backup_variables.txt` could be different from the output displayed during the `backup-and-apply-log` operation. (This issue did not occur if the backup and apply-log steps were done separately.) (Bug #16195529)
- A backup process might hang when it ran into an LSN mismatch between a data file and the redo log. This fix makes sure the process does not hang and it displays an error message showing the name of the problematic data file. (Bug #14791645)
- When using the `--only-innodb-with-frm` option, MySQL Enterprise Backup tried to create temporary files at unintended locations in the file system, which might cause a failure when, for example, the user had no write privilege for those locations. This fix makes sure the paths for the temporary files are correct. (Bug #14787324)

D.2 Changes in MySQL Enterprise Backup 3.8.1 (2013-02-05)

This section documents changes and bug fixes that have been applied in MySQL Enterprise Backup, from version 3.5.1 through version 3.8.1. The 3.8.1 release contains mainly fixes and enhancements compatibility with features of MySQL 5.6.

Functionality Added or Changed

- Prior to version 3.8.1, `mysqlbackup` ignores the `--slave-info` option if the `--no-locking` option was also used. `mysqlbackup` now throws an error if the two options are used together. (Bug #16582193)
- In MySQL 5.6, you can change the physical layout of the InnoDB system tablespace through the configuration options `innodb_undo_directory`, `innodb_undo_tablespaces`, and `innodb_undo_logs` (formerly `innodb_rollback_segments`). These options move the InnoDB undo log into one or more separate files, possibly outside the MySQL data directory or even on a different storage device. These files are named with sequential numbers: `undo001`, `undo002`, `undo003`, and so on, up to the number specified by `innodb_undo_tablespaces`.

  The `mysqlbackup` command recognizes these options during a backup, and retrieves the associated files from the specified location. When the `mysqlbackup` command creates a backup, it always stores these undo data files in the same directory as the other data files making up the system tablespace. During a restore operation, the files are copied to the appropriate location, depending on the setting for `innodb_undo_directory` on the server where restore takes place. (Bug #16168947)
- New server options present in MySQL 5.6 are accepted on the `mysqlbackup` command line when a database connection is not available, and are stored along with other instance-wide settings in the backup metadata:
  - The page size and checksum algorithm settings for the backed-up instance are recorded in the `backup_create.xml` file.
  - You can specify `--innodb_page_size=`*`SIZE`* and `--innodb_checksum_algorithm=`*`NAME`* on the command line if a database connection is not available.

  (Bug #16033618, Bug #15951510)
- MySQL Enterprise Backup can now handle InnoDB tables backed up to non-standard locations using the `DATA DIRECTORY` clause of the MySQL 5.6 `CREATE TABLE` statement. For each table created

outside the database subdirectory, a .isl file in the database subdirectory contains the absolute pathname of the associated .ibd file, acting like a symbolic link to the actual tablespace file.

The `mysqlbackup` command reads the `.isl` files to locate the associated `.ibd` files during a backup. It stores the `.isl` files in the backup directory using the extension `.bl`. If the server where you restore the backup does not have the same directory structure as the backup server, and you need to put the restored `.ibd` files in some different location, edit the `.bl` files before doing the restore. (Bug #15932375)

- Backups can be taken while online DDL operations are in progress on `InnoDB` tables. See InnoDB and Online DDL for details about this MySQL 5.6 feature. (Bug #15932180)
- MySQL Enterprise Backup supports the GTID feature of MySQL 5.6:
  - The GTID feature is compatible with the CSV tables that the `mysqlbackup` command uses to log job progress and history.
  - When a server using the GTID feature is backed up, `mysqlbackup` produces a file `gtid_executed.sql` as part of the backup data. This file contains a SQL statement that sets the `GTID_PURGED` configuration option. Execute this file using the `mysql` command line after restoring the backup data on a slave server. Optionally, you can uncomment the `CHANGE MASTER` command in this file and add any needed authentication parameters to it, before running it through `mysql`.
  - For servers not using GTIDs, you can use the `--slave-info` option as before, then edit and execute the `ibbackup_slave_info` file afterward.

  For more information about using MySQL Enterprise Backup with replication, see Chapter 6, *Using MySQL Enterprise Backup with Replication*. (Bug #14767426, Bug #14797808, Bug #14722659)
- MySQL 5.6 includes the `innodb_page_size` configuration option to set a page size other than the traditional 16KB default. The page size is set permanently when the instance is created, and cannot be changed afterward.

  The `mysqlbackup` command can back up MySQL instances that use any of the new page sizes (8KB, 4KB, 2KB, or 1KB). When restoring, set `innodb_page_size` to the same value as on the server that was backed up; otherwise, the restore operation halts with an error. (Bug #14761559, Bug #16070834)
- Normally, the InnoDB system tablespace is extended by one or more megabytes at a time. Under some circumstances, particularly in a disk-full or almost-full situation, MySQL can extend the InnoDB system tablespace by some amount less than a megabyte, and this trailing portion is unused. Now the `mysqlbackup` command backs up such oddly sized system tablespaces even though the size does not precisely match the specified size. This condition causes a warning rather than an error. The unused fractional megabyte is removed from the backup data. (This particular file-size feature applies only to the system tablespace, not to the file-per-table tablespaces in the .ibd files.)
- The `mysqlbackup` command now supports the UNC (Universal or Uniform Naming Convention) syntax for specifying locations on shared disks. This feature lets you start backups to shared drives using Windows Task Scheduler, when shared drives cannot be mapped to a drive letter.

  The UNC syntax for Windows systems has the generic form:

```
\\ComputerName\SharedFolder\Resource
```

  You can use either forward or backward slashes in the path names.

  The `mysqlbackup` command does not support the "long UNC" syntax:

```
\\?\UNC\ComputerName\SharedFolder\Resource
```

The UNC path names can be specified with any backup command and option.

> **Note**
>
> Do not use UNC paths to specify the location of InnoDB data files and log files to be backed up. Such files cannot be reliably backed up over a network file system. MySQL Enterprise Backup does not issue any warning in this case.

Bugs Fixed

- **Important Change:** When backing up a slave server using the `--slave-info` option, updates could be lost when restoring the backup and starting replication using the command in the `meta/ibbackup_slave_info` file. This issue could arise if the replication SQL thread was not in synch with the replication I/O thread.

  Prior to this fix, the workaround for backups with the `--slave-info` option was to stop the I/O thread and wait for the SQL thread to catch up before starting the backup. (Bug #13588571, Bug #15865869)

- An apply-log operation could fail if the server used an `innodb_page_size` setting different from the default of 16KB. (Bug #16084912)

- The `--suspend-at-end` and `--exec-when-locked` options did not work properly in MySQL Enterprise Backup 3.8. The suspension code has been rewritten to work in conjunction with the multithreaded system of readers and writers. (Bug #16078126, Bug #16168131)

- MySQL Enterprise Backup 3.8.1 now supports the different checksum settings of the `innodb_checksum_algorithm` configuration option in MySQL 5.6.

  Prior to MySQL Enterprise Backup 3.8.1, when the value of the `innodb_checksum_algorithm` was `strict_crc32` or `strict_none`, the server could not be started after a sequence of backup, apply-log, and restore operations. Checksums were created in the InnoDB data files during the apply-log phase that were incompatible with the checksum algorithms required in the server.

  With this fix in place, there are still some restrictions regarding the settings for the `innodb_checksum_algorithm` during backup and restore:

  - A server backed up while the `strict_crc32`, `strict_innodb`, or `strict_none` setting is in effect for `innodb_checksum_algorithm` must be restored to a server with that same `innodb_checksum_algorithm` setting.
  - If a server contains data produced under different `innodb_checksum_algorithm` settings, because the option setting was changed during the lifetime of the server, that data cannot be restored to a server with one of the `strict_*` options for `innodb_checksum_algorithm`.

  (Bug #15951510)

- When a slave server was backed up with the `--slave-info` option, the meta file `backup_content.xml` contained incorrect data in the section underneath the `<binlog_file>` tag. (Bug #15926218)

- The `mysqlbackup` command could encounter a severe error during the `apply-log` step, or the corresponding phase of `apply-incremental-backup`, if DML or DDL had been done during the backup. (The apply log algorithm rounded the size of data files to the next lower 1MB border, if the size was not an exact number of megabytes. The apply log algorithm also did not ignore temporary data files and failed when it detected duplicate tablespace IDs.) (Bug #15841875, Bug #14704347, Bug #15932180)

- A backup operation could fail with a serious error (segmentation fault) if the `--connect-if-online` option was specified and the MySQL server was not online. (Bug #14788375)

- There are 2 database connections used by the `mysqlbackup` command. The first connection is used for reading server configuration, and locking and updating the history table. The second connection is used for reading incremental base details during incremental backup. The second connection did not specify an `autocommit` setting, possibly causing a stall when the server had the setting `autocommit=0`. Now `mysqlbackup` specifies `autocommit=1` automatically in this connection. (Bug #14637052)

- If the .ibd file for an `InnoDB` table created under the `innodb_file_per_table` option was renamed or removed while a backup was in progress, the `mysqlbackup` command could fail. This issue primarily affected tables with a size greater than 60 MB that were dropped or renamed during the backup operation. (Bug #14590007)

- The password specified by the `--password` option was printed in `mysqlbackup` command output, rather than being masked. This issue affected the `mysqlbackup` command, but any password parameters recorded in the `backup_history` table were masked correctly. (Bug #14576054)

- The default for the option `--only-innodb-with-frm` was mistakenly set to `all` when it was intended to be `related`. The default for that option is now `related`. (Bug #14507779)

- This fix addresses problems doing a single-file backup (using the `--backup-image` option) when the `innodb_data_file_path` or `innodb_data_home_dir` configuration options contained absolute paths. This issue typically affected configurations with `InnoDB` files split across multiple storage devices. The symptoms and associated fixes are:

  - The backup operation failed if `InnoDB` data files were placed in a subdirectory of `datadir`. Now `mysqlbackup` creates all required subdirectories in the `backup_dir` directory.

  - The backup operation failed if `InnoDB` data files were placed in a sub-subdirectory of `datadir`. It refused to work if any directory was found in the database directories. The error is now a warning instead.

  - `InnoDB` data files were copied twice, if they were in a subdirectory of `datadir`: once during the phase the copies `InnoDB` files and again in the phase that copies non-`InnoDB` files. During this second phase, all files in the database directories are copied, which mistakenly included the `InnoDB` files from subdirectories of `datadir`. Now `mysqlbackup` compares each filename from the `datadir` subdirectories against each `InnoDB` data file name from the system tablespace, to avoid copying the files a second time. .ibd files were not affected by this issue, as they are recognized by their file name extension.

  - MEB can now back up system tablespace data files inside the source `datadir` if `backup_innodb_data_home_dir` and `backup_innodb_data_file_path` were configured accordingly. Fixed by checking each backup target system tablespace data file against the server's `datadir`.

  (Bug #14499912)

- On a busy server with a high write workload, the `mysqlbackup` command could fail with an error such as:

  ```
  mysqlbackup: ERROR: Page at offset 0 number in
  /path/db_name#table_name.ibd seems corrupt!
  ```

  The problem was due to a race condition where a checksum was written just as the page was being backed up, before the corresponding data was stored on the page. Now `mysqlbackup` waits and re-tries the read operation when a checksum mismatch occurs. The new `mysqlbackup` options `--page-reread-time` and `--page-reread-count` let you adjust the mechanism for re-trying page reads. If you continue to encounter such errors, re-try the backup with higher values for these options to reduce the possibility of encountering the race condition. (Bug #14489495)

- A configuration file with `--ssl-*` options in the `[client]` section caused the `mysqlbackup` command to fail, rather than being ignored as with most other unused options. A workaround was to copy the configuration file, remove the `--ssl-*` parameters, and run `mysqlbackup` with the `--defaults-file` option to use the smaller configuration file.

  The fix causes `mysqlbackup` to recognize and handle all the SSL options currently recognized by the server: `--ssl`, `--ssl-key`, `--ssl-cert`, `--ssl-ca`, `--ssl-capath`, `--ssl-cipher`, and `--ssl-verify-server-cert`. (Bug #13520946)

- The `--slave-info` option is not compatible with `--only-innodb-with-frm`, `--only-innodb-with-frm=all`, or `--only-innodb-with-frm=related`. Now the `mysqlbackup` command reports a command-line error if you specify those options in combination. (Bug #13414742, Bug #63355)

- If the MySQL server had a low value for the `wait_timeout` configuration option, large backups (particularly for large `MyISAM` tables) could fail due to timeout errors. The timeout could also prevent the `mysql.backup_progress` table from recording the final details of the failed backup. The fix raises the value of `wait_timeout` to the maximum value within the session opened by the `mysqlbackup` command. The maximum timeout value is platform-specific: approximately 24 days on Windows systems, and 1 year for other platforms. (Bug #13258784, Bug #13416025, Bug #14468879)

D.3 Changes in MySQL Enterprise Backup 3.8.0 (2012-07-27)

This section documents changes and bug fixes that have been applied in MySQL Enterprise Backup, from version 3.5.1 through version 3.8.0. The primary feature of MySQL Enterprise Backup 3.8.0 is performance-related: parallel backup and its associated options `--number-of-buffers`, `--read-threads`, `--write-threads`, and `--process-threads`.

Functionality Added or Changed

- **Performance:** During both backup and restore jobs, certain operations are performed in parallel: reading data, writing data, compressing or decompressing it (if applicable), and validating it (verifying the page checksum during backup). The parallel execution takes advantage of multi-threading. The performance increase is particularly evident when backing up to or restoring from a RAID storage device, and when using compressed backups. See Section 4.1.11, “Performance / Scalability / Capacity Options” for the options to fine-tune the parallel execution parameters. See Chapter 7, *Performance Considerations for MySQL Enterprise Backup* for general performance advice, including tips for parallel processing for backup and restore.

Bugs Fixed

- A backup using the combination of options `--incremental-with-redo-log-only` and `--incremental-base=history:last_backup` could crash the `mysqlbackup` command. The issue was intermittent, depending on the timing of updates to the database and the beginning of the incremental backup. The workaround was to re-run the incremental backup operation after a delay of a few seconds. (Bug #14241403)

- The output of the `mysqlbackup` command was enhanced to include additional timestamp and summaries of data volume and throughput. (Bug #14054071)

- An error could occur when running the apply-log phase on a single-file backup image on a Linux system, if the backup was originally performed on a Windows system.i The cause was a mismatch between the OS-specific file separator characters, resulting in the image file being unpacked into a set of files in a single directory, rather than into a directory tree. The error messages looked like:

  ```
  InnoDB: Operating system error number 2 in a file operation.
  InnoDB: The error means the system cannot find the path specified.
  ```

  (Bug #13477633)

- The metadata file `backup_content.xml` could list files with names matching the pattern `ibdata*`, for example `ibdata1` and `ibdata2`, that did not exist in the backup. The information in the metadata file is for informational purposes only, so the extra filenames did not cause any issues with the backup or restore procedures. (Bug #12554059)
- For a single-file backup created using the `backup-to-image` or `backup-dir-to-image` option, the metadata file `backup_content.xml` could show file sizes for `ibdata*` files that did not match the sizes of the actual files in the backup. The mismatch occurred because huge files are divided into pieces when stored in the single-file backup image. The information in the metadata file is for informational purposes only, so the mismatching file sizes did not cause any issues with the backup or restore procedures. (Bug #12549658)
- Backups using the `--include` option included all tables, rather than only those included in the regular expression argument. (Bug #11772424)

D.4 Changes in MySQL Enterprise Backup 3.7.1 (2012-03-23)

This section documents changes and bug fixes that have been applied in MySQL Enterprise Backup, from version 3.5.1 through version 3.7.1. MySQL Enterprise Backup 3.7.1 is primarily a bug-fix release, with one new feature: the `--incremental-base=history:last_backup` option syntax to simplify taking a sequence of incremental backups.

Functionality Added or Changed

- New argument syntax for the `--incremental-base` option makes it simpler to perform a sequence of incremental backups. When you specify the option combination `--incremental --incremental-base=history:last_backup`, the `mysqlbackup` command uses the metadata in the `mysql.backup_history` table to determine the LSN to use as the lower limit of the incremental backup. You no longer need to keep track of the actual LSN (as in the option `--start-lsn=`*`LSN`*) or even the location of the previous backup (as in the option `--incremental-base=dir:`*`directory_path`*). (Bug #11765316)
- The `ibbackup` and `innobackup` commands, provided for compatibility with command syntax from earlier releases are deprecated, meaning that they could be removed in a future release. For more information, see Appendix B, *Compatibility Information for MySQL Enterprise Backup*.

Bugs Fixed

- **Important Change:** When backing up a slave server using the `--slave-info` option, updates could be lost when restoring the backup and starting replication using the command in the `meta/ibbackup_slave_info` file. This issue could arise if the replication SQL thread was not in synch with the replication I/O thread.

  Prior to this fix, the workaround for backups with the `--slave-info` option was to stop the I/O thread and wait for the SQL thread to catch up before starting the backup. (Bug #13588571, Bug #15865869)
- An error (segmentation fault) could occur when using the Tivoli Storage Manager product for backups, in combination with the SBT interface and associated `--sbt*` options for `mysqlbackup`. (Bug #13738674)
- A single-file backup produced on a Linux system could not be extracted on a Windows system. The `mysqlbackup` command with the `image-to-backup-dir` option would fail with the message:

  ```
  mysqlbackup.exe: ERROR: Could not extract contents of image file.
  ```

  (Bug #13478310)
- When `mysqlbackup` was run with the `--only-innodb-with-frm` option, it incorrectly displayed messages indicating that tables were being locked. The tables were not actually being locked. The fix suppresses the erroneous messages. (Bug #13477573)

- If the MySQL server had a low value for the `wait_timeout` configuration option, large backups (particularly for large `MyISAM` tables) could fail due to timeout errors. The timeout could also prevent the `mysql.backup_progress` table from recording the final details of the failed backup. The fix raises the value of `wait_timeout` to the maximum value within the session opened by the `mysqlbackup` command. The maximum timeout value is platform-specific: approximately 24 days on Windows systems, and 1 year for other platforms. (Bug #13258784, Bug #13416025, Bug #14468879)

- The operation of the `--force` [45] operation has been fine-tuned and clarified.

  Now `--force` allows overwriting InnoDB data and log files in combination with the `apply-log` and `apply-incremental-backup` options, and replacing the image file in combination with the `backup-to-image` and `backup-dir-to-image` options. For all other mode options, the `--force` option is rejected.

  It is now possible to perform another `apply-log` and `apply-incremental-backup`, even after a successful apply operation. Previously, this was not allowed by `mysqlbackup`. (Bug #12922426)

- The file `meta/image_files.xml` was incorrectly displayed twice when listing the contents of a single-file backup. (Bug #12563349)

- Specifying a very large memory limit, such as `--limit-memory=999999`, could cause a segmentation fault when running the `mysqlbackup` command with the `apply-log` option. (Bug #11779864)

- The `engines` column in the `mysql.backup_history` table does not correctly reflect the storage engines of the backed-up databases. This has been documented as a current limitation. (Bug #11763818, Bug #56582)

- When `mysqlbackup` is run with the `copy-back` option, now it displays warnings as existing files are overwritten. (Bug #11760714)

D.5 Changes in MySQL Enterprise Backup 3.7.0 (2012-01-04)

This section documents changes and bug fixes that have been applied in MySQL Enterprise Backup, from version 3.5.1 through version 3.7.0.

Functionality Added or Changed

- The single-file backup feature now includes checksum verification to ensure the backup data remains unchanged during any transfers to other systems. Each file within the backup image is tested against a checksum calculated using the CRC32 algorithm, either when files are extracted from the backup image, or using the new `mysqlbackup` option `validate` to test a backup image without extracting. For example:

  ```
  mysqlbackup other_options --backup-image=image_file validate
  ```

- The `mysqlbackup` option `--start-lsn` is now optional for incremental backups. If you specify the location of an existing full or incremental backup with the new `--incremental-base` option, you can omit the `--start-lsn` option and the `mysqlbackup` command automatically determines the appropriate LSN.

  For safety, the backup job halts with an error if the LSN information cannot be retrieved, or if there is a mismatch between the LSN recorded in the previous backup and the LSN reported by the MySQL instance.

  The automatic LSN detection works with both the original (`--incremental`) and new (`--incremental-with-redo-log-only`) incremental backup techniques.

- The `mysqlbackup` now offers an alternative method of performing incremental backup for InnoDB tables. This technique involves copying the redo log. This backup method is an alternative to the incremental backup of InnoDB tables.

  To enable the redo-log-only incremental backup method, specify the `--incremental-with-redo-log-only` option on the `mysqlbackup` command line, instead of the `--incremental` option).

  > **Note**
  >
  > The benefit of this option is a reduction in random I/O during the backup stage, in favor of sequential I/O. The older incremental backup technique reads only changed blocks from the InnoDB data files. If a high proportion of blocks changed, I/O overhead to read the changed data could limit the performance benefits of an incremental backup. The new technique minimizes this I/O overhead by doing sequential reads to copy the relatively small `ib_logfile*`files. The new technique tends to perform better on systems with big InnoDB tables that receive frequent updates, broadly distributed within the tables. If you have small tables, or relatively few changes, or changes over and over again to the same pages on disk, the old technique might perform better for you. The backup information from the redo logs might also take longer to apply during the apply-log phase. Run benchmarks to determine the best technique for your workload.
  >
  > This technique relies on you having sufficient redo log data to cover the period since the last incremental backup. Because the InnoDB redo log uses a circular buffer where older entries eventually are overwritten, make incremental backups frequently to avoid having older changes unavailable for the incremental backup. Ensure that the value of the `innodb_log_file_size` configuration option is large enough to hold all the changes to InnoDB data that accumulate between incremental backups.
  >
  > When you apply the log data from a series incremental backups to a full backup, the incremental backup data can be produced using either of the techniques. Some incremental backups could be produced by the `--incremental` option and others with the `--incremental-with-redo-log-only` option.
  >
  > Because the incremental backup data format produced by the `--incremental-with-redo-log-only` option is different from the format of MySQL Enterprise Backup 3.6 or 3.5, older `mysqlbackup` versions cannot perform the apply step on a redo-log-only backup. If you use the new incremental backup technique, upgrade the `mysqlbackup` command to the latest MySQL Enterprise Backup level on any machines where you run the apply-log step.
  >
  > This feature is not available in the `ibbackup` command, which is intended only for compatibility with the option syntax from MySQL Enterprise Backup 3.5 and the earlier InnoDB Hot Backup product.

- Performance work within the `mysqlbackup` command makes backup jobs faster with less overhead.

  Performance of backup-related I/O operations is improved, particularly on Windows, by reusing I/O library code and best practices from the MySQL Server product.

  CPU overhead is lessened by reducing the number of memory allocation and deallocation operations.

- A new option for the `mysqlbackup`, `--only-innodb-with-frm`, lets you back up InnoDB tables and their associated `.frm` files with minimal disruption to database processing.

By default, all InnoDB and non-InnoDB tables are backed up, along with all the `.frm` files. While the `.frm` files are being copied, the instance is put into a read-only state. With the `--only-innodb` option introduced in MySQL Enterprise Backup 3.6, only InnoDB tables are backed up, but you must copy the `.frm` files manually, and again this stage happens with the instance in a read-only state. The `--only-innodb-with-frm` is intended for backups where you can ensure that no `ALTER TABLE`, `CREATE TABLE`, `DROP TABLE`, or other DDL statements modify the `.frm` files for InnoDB tables during the backup operation. If the `mysqlbackup` detects that any of the relevant `.frm` files was modified or deleted during the backup job, the command halts with an error.

- When managing backup data using the SBT interface of products such as Oracle Secure Backup, you can change certain settings in the media management software by setting environment variables that are recognized by the SBT library. The new `--sbt-environment` option of the `mysqlbackup` command lets you set such environment variables for the duration of the backup job only, rather than using a wrapper script to set and unset the variables.

  Although the intended purpose of the `--sbt-environment` option is to pass environment variables used by the SBT library, you can set any Unix, Linux, or Windows environment setting this way for the duration of the backup job.

Bugs Fixed

- The `backup` and some combinations of `extract` and `--src-entry` options could fail depending on the existence or non-existence of target directories. In most cases, the error was due to a directory not existing and `mysqlbackup` failing to create it. An `extract` option with no `--src-entry` option could fail if the target directory did already exist. (Bug #13465782)
- When using `mysqlbackup` with the `copy-back` option, specifying a `--datadir` option with a trailing slash on the directory name could fail with an uninformative error message. This issue only occurred on Windows systems. (Bug #13392053)
- A backup operation could fail if the database being backed up contained only views and no tables. (Bug #13359833)
- MySQL Enterprise Backup now does stricter checking of LSN values to prevent the apply step for incremental backups to be carried out in an incorrect order. Now `mysqlbackup` gives an error if asked to apply an incremental backup that is older than the full backup, or if another incremental backup needs to be applied first. (Bug #13350012)
- Attempting to re-run an `apply-log` operation a second time would fail, when using the `--force` [45] option to overwrite files. (Bug #13012586)
- A missing `datadir` option in the MySQL configuration file could cause the `mysqlbackup` command to halt. (Bug #12838474)
- During a backup using the `--only-innodb` option, an incorrect message was displayed regarding a time interval to copy non-InnoDB files. (Bug #12691358)
- Fixed a potential syntax error in the `CHANGE MASTER` statement written to the `ibbackup_slave_info` file by the `--slave-info` option. (Bug #12540081)
- The `apply-log` and `copy-back` options now print messages showing the times for the start and end of those operations. (Bug #12313355, Bug #12837622)
- Clarified processing of duplicate options. When the same option is specified more than once, the last one takes precedence. This is standard MySQL practice, not an error or warning situation. (Bug #11763378, Bug #56076)

D.6 Changes in MySQL Enterprise Backup 3.6.1 (2011-09-28)

Functionality Added or Changed

- MySQL Enterprise Backup can now authenticate to the server being backed up using the Enterprise authentication plugins available in the commercial distributions for MySQL 5.5.16 and higher. For example:

  - With the Windows Native authentication plugin, you can set up a MySQL user ID named the same as the Windows user ID, grant MySQL privileges as described in Section 3.1.2, “Grant MySQL Privileges to Backup Administrator”, and then perform backups from that Windows account by specifying the `--user` option without a `--password` option.

  - With the PAM authentication plugin, you can connect to the MySQL server using a flexible system to map user IDs and associated privileges.

  For more details about the MySQL pluggable authentication feature, see Pluggable Authentication.

Bugs Fixed

- Under some circumstances, the `mysqlbackup` with the `--no-locking` option halted with the message `Backup of non-innodb tables failed`. Now, the `--no-locking` option prevents this issue. (Bug #12952150)

- Under MySQL 5.5.8 and higher, a full backup using the `mysqlbackup` command could fail with the combination of settings `binlog_format=ROW` and `transaction-isolation=READ-COMMITTED`. The error message was:

  ```
  mysqlbackup: ERROR: Could not lock tables. Aborting.
  mysqlbackup: ERROR: Backup of non-innodb tables failed.!
  ```

  (Bug #12922167, Bug #62268)

- Specifying `mysqlbackup` options incorrectly could cause a fatal error. For example, using an underscore (`apply_log`) instead of a dash (`apply-log`), or misspelling an option (for example, `copy-back`), caused `mysqlbackup` to halt. Now, incorrect options produce a descriptive error message rather than an assertion failure. (Bug #12780833)

- The `mysqlbackup` options `copy-back`, `apply-log`, and `apply-incremental-backup` did not print the success message “mysqlbackup completed OK!”, even when the operation was successful. (Bug #12710941)

- This fix changes the way non-InnoDB files are handled when applying an incremental backup to a full backup. The behavior differs depending on whether or not the incremental backup was taken with the `--only-innodb` option.

  In MySQL Enterprise Backup 3.5, when applying an incremental backup, `.frm` files were deleted from the full backup, if they were not present in the incremental backup. In MySQL Enterprise Backup 3.6.0, this behavior changed, so that applying an incremental backup to a full backup would never delete `.frm` and other non-InnoDB files. This change made it more convenient to take a full backup, followed by incremental backups of InnoDB tables using the `--only-innodb` option. But if a table was dropped, its `.frm` file would not be removed when subsequent incremental backups were taken and applied to the full backup. The table would be reported by the `SHOW TABLES` statement, but would give an error when accessed by SQL statements.

  With this bug fix, an incremental backup using default options reverts to the original behavior, synchronizing the `.frm` files with the full backup, including deleting them when appropriate. Incremental backups with the `--only-innodb` option retain the cautious behavior that never deletes `.frm` and other non-InnoDB files when applied to full backups. If you use `--only-innodb` with incremental backups, you must handle the deletion of non-InnoDB files yourself in the full backup directory. (Bug #12636719)

- The `backup-to-image` option to produce a single-file backup left behind zero-byte temporary files `ibdata1` and `mysql/ibbackup_binlog_marker.ibd` after completing. These files were left

behind in the work directory specified by the `--backup-dir` option, and in the image file. Now these files are removed as intended. (Bug #12408255)

D.7 Changes in MySQL Enterprise Backup 3.6.0 (2011-07-01)

This section documents changes and bug fixes that have been applied in MySQL Enterprise Backup, version 3.6. This release has substantial enhancements to `mysqlbackup` syntax and processing over MySQL Enterprise Backup 3.5 and the older InnoDB Hot Backup product. For details, see Appendix B, *Compatibility Information for MySQL Enterprise Backup*.

Functionality Added or Changed

- The `mysqlbackup` command gains enhanced capabilities to do cold backups, with the `--connect-if-online` option.
- The `mysqlbackup` command can now interface with Media Management Software (MMS) products such as Oracle Secure Backup, using the System Backup to Tape (SBT) protocol.
- The backup operation now is much more “online” than in the past.

  Several new options specify connection information and credentials for the database being backed up.

  The connection-related options are made consistent with the corresponding options used for other MySQL client programs.

  You no longer need to construct a dummy configuration file for use with MySQL Enterprise Backup. The `mysqlbackup` command reads options from the standard MySQL configuration file, either from its own `[mysqlbackup]` group or the generic `[client]` group. Details about the layout and locations of files in the MySQL server are retrieved automatically using the database connection, so that you do not need to specify them in the configuration file.
- For simplicity in managing and transferring backup data, you can produce a single-file backup as an alternative to a directory tree of backup files. The single-file backup is a foundational feature that is the basis for other important MySQL Enterprise Backup capabilities, such as streaming the backup data to another server and managing the backup data through a Media Management Software product such as Oracle Secure Backup.
- A new `meta` subdirectory inside the backup data contains information about the backup itself. This metadata is known collectively as the manifest. You can use this information to build additional reporting or management features on top of MySQL Enterprise Backup.
- You can associate comments with each set of backup data, either a single string specified on the command line, or through a separate text file.
- For the fastest backup with the least disruption to MySQL server processing, options such as `--innodb-only` and`--no-locking` let you back up InnoDB tables exclusively. By skipping the backup of non-InnoDB files such as MyISAM tables and `.frm` files, you can avoid the final phase of the backup that waits for other operations in the server to complete, then puts the server into a read-only state.

Bugs Fixed

- The `mysqlbackup` command could fail when the size of the `ibbackup_logfile` file in the backup directory exceeded 4GB. (Bug #12590463)
- Fixed a potential syntax error in the `CHANGE MASTER` statement written to the `ibbackup_slave_info` file by the `--slave-info` option. (Bug #12540081)
- When applying the log to a compressed backup, the operation could crash if the `--uncompress` option was omitted. Now, instead of the crash, an error message is displayed about the required option. (Bug #11780068)

- Documented the maximum number of subdirectories (21) allowed in the `backup-dir` path. (Bug #11766768, Bug #59958)
- If the MySQL server was running with the setting `SQL_MODE='TRADITIONAL'`, the `mysqlbackup` command could not create the `backup_history` table. This was a minor issue that did not halt the backup operation. (Bug #11766646, Bug #59800)
- The `mysqlbackup` command could crash during the apply-log stage when a database was dropped between a full backup and a subsequent incremental backup. (Bug #11766499, Bug #59623)
- The `mysqlbackup` command could fail on Windows systems if the path to the MySQL configuration file contained spaces. (Bug #11764927, Bug #57824)

D.8 Changes in MySQL Enterprise Backup 3.5.4 (2011-04-21)

This section documents changes and bug fixes that have been applied in MySQL Enterprise Backup, version 3.5.4.

Bugs Fixed

- The apply-log operation for an incremental backup could fail on Windows with error similar to:

```
110406  9:43:23  InnoDB: Operating system error number 0 in a file operation.
...
ibbackup: Error: cannot delete
```

  (Bug #12328828)

- If an error occurred during a backup, the `start_time` and `end_time` of the backup run could be incorrect in the `backup_history` table. (Bug #11900590)
- When an incremental backup was taken of a database using per-table tablespaces, while `ALTER TABLE` statements were running, the apply-log phase could fail, leaving the full backup in an inconsistent state. (Bug #11766088, Bug #59126)
- Running an incremental backup on a database with per-table tablespaces could fail on Windows systems. (Bug #11765740, Bug #58734)
- A blank value for the `innodb_data_home_dir` configuration option would cause the `ibbackup` command to fail. This fix allows you to specify multiple directory names in the `innodb_data_file` configuration option and specify `innodb_data_home_dir` with a blank value. (Bug #59394, Bug #11766307)
- For a system where the LSN has reached a value exceeding 2**31, an incremental backup could fail with the error message:

```
mysqlbackup: Error: --incremental is given but --lsn is not or wrong value
```

  (Bug #59090)

- Minor fixes for copyright notices.

D.9 Changes in MySQL Enterprise Backup 3.5.2 (2010-12-16)

This section documents changes and bug fixes that have been applied in MySQL Enterprise Backup, version 3.5.2.

Functionality Added or Changed

- A call to `posix_fadvise()` can be used to reduce the flush cycle of the operating system cache and improve backup performance. This option is set on by default.

- The combined InnoDB and MyISAM backup functionality of the `innobackup` command is now available on Windows systems. The former Perl script is rewritten in C/C++ as the `mysqlbackup` command. This release continues to include the `innobackup` command, which may be deprecated by the next release. There are also some changes to the syntax as specified in the manual.

- Backup history and progress information is logged to the `mysql.backup_history` and `mysql.backup_progress` tables, so that it can be used by the MySQL Enterprise Monitor product and other tools to easily monitor backup operations. For the details of the backup history table, see Chapter 9, *Troubleshooting for MySQL Enterprise Backup*.

Bugs Fixed

- The apply-log step for an incremental backup would fail if the `innodb_log_group_home_dir` and `datadir` values specified in the configuration file were not the same. (Bug #57375)

- The file `ibbackup_binlog_info` in the backup directory is now updated when an incremental backup is applied, to reflect the updated binlog position and LSN of the full backup. (Bug #57286)

D.10 Changes in MySQL Enterprise Backup 3.5.1 (2010-11-01)

This section documents changes and bug fixes that have been applied in MySQL Enterprise Backup, version 3.5.1.

Functionality Added or Changed

- Incremental backup.

- Support for the Barracuda file format of InnoDB. MySQL Enterprise Backup can now backup tables that use recent InnoDB features such as table compression and the `dynamic` row format.

Bugs Fixed

- The `innobackup` or `mysqlbackup` command could create an orphaned table in the backup directory. The file `mysql/ibbackup_binlog_marker.ibd` was created in the backup directory, but not `mysql/ibbackup_binlog_marker.frm`. The resulting table `mysql.ibbackup_binlog_marker` could not be dropped or re-created, which could prevent subsequent backups from succeeding. This condition could occur when a partial backup was created with the `--databases` option, and the database had multiple tablespaces from the setting `--innodb-file-per-table=1`. Now, the `.frm` file for this internally produced table is copied into the backup without the table being specified as part of the `--databases` argument list. (Bug #54454)

Appendix E Licenses for Third-Party Components

Table of Contents

Oracle acknowledges that certain Third Party and Open Source software has been used to develop or is incorporated in the MySQL Enterprise Backup product. This appendix includes required third-party license information.

E.1 RegEX-Spencer Library License

The following software may be included in this product:

```
Henry Spencer's Regular-Expression Library (RegEX-Spencer)

Copyright 1992, 1993, 1994, 1997 Henry Spencer. All rights reserved.
This software is not subject to any license of the American Telephone
and Telegraph Company or of the Regents of the University of
California.

Permission is granted to anyone to use this software for any purpose
on any computer system, and to alter it and redistribute it, subject
to the following restrictions:

1. The author is not responsible for the consequences of use of this
software, no matter how awful, even if they arise from flaws in it.

2. The origin of this software must not be misrepresented, either by
explicit claim or by omission. Since few users ever read sources,
credits must appear in the documentation.

3. Altered versions must be plainly marked as such, and must not be
misrepresented as being the original software.  Since few users ever
read sources, credits must appear in the documentation.

4. This notice may not be removed or altered.
```

E.2 zlib License

The following software may be included in this product:

zlib

Oracle gratefully acknowledges the contributions of Jean-loup Gailly and Mark Adler in creating the zlib general purpose compression library which is used in this product.

```
zlib.h -- interface of the 'zlib' general purpose compression library
Copyright (C) 1995-2004 Jean-loup Gailly and Mark Adler

zlib.h -- interface of the 'zlib' general purpose compression library
version 1.2.3, July 18th, 2005
Copyright (C) 1995-2005 Jean-loup Gailly and Mark Adler

zlib.h -- interface of the 'zlib' general purpose compression library
version 1.2.5, April 19th, 2010
```

```
Copyright (C) 1995-2010 Jean-loup Gailly and Mark Adler

This software is provided 'as-is', without any express or implied warranty.
In no event will the authors be held liable for any damages arising from the
use of this software. Permission is granted to anyone to use this software
for any purpose,including commercial applications, and to alter it and
redistribute it freely, subject to the following restrictions:

1. The origin of this software must not be misrepresented; you must not
   claim that you wrote the original software. If you use this software
   in a product, an acknowledgment in the product documentation would
   be appreciated but is not required.
2. Altered source versions must be plainly marked as such, and must not
   be misrepresented as being the original software.
3. This notice may not be removed or altered from any source distribution.

Jean-loup Gailly jloup@gzip.org
Mark Adler madler@alumni.caltech.edu
```

E.3 Percona Multiple I/O Threads Patch License

The following software may be included in this product:

Percona Multiple I/O threads patch

```
Copyright (c) 2008, 2009 Percona Inc
All rights reserved.

Redistribution and use of this software in source and binary forms,
with or without modification, are permitted provided that the
following conditions are met:

 * Redistributions of source code must retain the above copyright
   notice, this list of conditions and the following disclaimer.
 * Redistributions in binary form must reproduce the above copyright
   notice, this list of conditions and the following disclaimer in the
   documentation and/or other materials provided with the distribution.
 * Neither the name of Percona Inc. nor the names of its contributors
   may be used to endorse or promote products derived from this software
   without specific prior written permission of Percona Inc.

THIS SOFTWARE IS PROVIDED BY THE COPYRIGHT HOLDERS AND CONTRIBUTORS
"AS IS" AND ANY EXPRESS OR IMPLIED WARRANTIES, INCLUDING, BUT NOT
LIMITED TO, THE IMPLIED WARRANTIES OF MERCHANTABILITY AND FITNESS
FOR A PARTICULAR PURPOSE ARE DISCLAIMED. IN NO EVENT SHALL THE
COPYRIGHT OWNER OR CONTRIBUTORS BE LIABLE FOR ANY DIRECT, INDIRECT,
INCIDENTAL, SPECIAL, EXEMPLARY, OR CONSEQUENTIAL DAMAGES (INCLUDING,
BUT NOT LIMITED TO, PROCUREMENT OF SUBSTITUTE GOODS OR SERVICES;
LOSS OF USE, DATA, OR PROFITS; OR BUSINESS INTERRUPTION) HOWEVER
CAUSED AND ON ANY THEORY OF LIABILITY, WHETHER IN CONTRACT, STRICT
LIABILITY, OR TORT (INCLUDING NEGLIGENCE OR OTHERWISE) ARISING IN
ANY WAY OUT OF THE USE OF THIS SOFTWARE, EVEN IF ADVISED OF THE
POSSIBILITY OF SUCH DAMAGE.
```

E.4 Google SMP Patch License

The following software may be included in this product:

Google SMP Patch

```
Google SMP patch

Copyright (c) 2008, Google Inc.
All rights reserved.

Redistribution and use in source and binary forms, with or without
modification, are permitted provided that the following conditions
```

```
are met:

 * Redistributions of source code must retain the above copyright
   notice, this list of conditions and the following disclaimer.
 * Redistributions in binary form must reproduce the above copyright
   notice, this list of conditions and the following disclaimer in the
   documentation and/or other materials provided with the distribution.
 * Neither the name of the Google Inc. nor the names of its contributors
   may be used to endorse or promote products derived from this software
   without specific prior written permission.

THIS SOFTWARE IS PROVIDED BY THE COPYRIGHT HOLDERS AND CONTRIBUTORS
"AS IS" AND ANY EXPRESS OR IMPLIED WARRANTIES, INCLUDING, BUT NOT
LIMITED TO, THE IMPLIED WARRANTIES OF MERCHANTABILITY AND FITNESS
FOR A PARTICULAR PURPOSE ARE DISCLAIMED. IN NO EVENT SHALL THE
COPYRIGHT OWNER OR CONTRIBUTORS BE LIABLE FOR ANY DIRECT, INDIRECT,
INCIDENTAL, SPECIAL, EXEMPLARY, OR CONSEQUENTIAL DAMAGES (INCLUDING,
BUT NOT LIMITED TO, PROCUREMENT OF SUBSTITUTE GOODS OR SERVICES;
LOSS OF USE, DATA, OR PROFITS; OR BUSINESS INTERRUPTION) HOWEVER
CAUSED AND ON ANY THEORY OF LIABILITY, WHETHER IN CONTRACT, STRICT
LIABILITY, OR TORT (INCLUDING NEGLIGENCE OR OTHERWISE) ARISING IN
ANY WAY OUT OF THE USE OF THIS SOFTWARE, EVEN IF ADVISED OF THE
POSSIBILITY OF SUCH DAMAGE.
```

E.5 Google Controlling Master Thread I/O Rate Patch License

The following software may be included in this product:

Google Controlling master thread I/O rate patch

```
Copyright (c) 2009, Google Inc.
All rights reserved.

Redistribution and use in source and binary forms, with or without
modification, are permitted provided that the following conditions
are met:

 * Redistributions of source code must retain the above copyright
   notice, this list of conditions and the following disclaimer.
 * Redistributions in binary form must reproduce the above copyright
   notice, this list of conditions and the following disclaimer in the
   documentation and/or other materials provided with the distribution.
 * Neither the name of the Google Inc. nor the names of its contributors
   may be used to endorse or promote products derived from this software
   without specific prior written permission.

THIS SOFTWARE IS PROVIDED BY THE COPYRIGHT HOLDERS AND CONTRIBUTORS
"AS IS" AND ANY EXPRESS OR IMPLIED WARRANTIES, INCLUDING, BUT NOT
LIMITED TO, THE IMPLIED WARRANTIES OF MERCHANTABILITY AND FITNESS
FOR A PARTICULAR PURPOSE ARE DISCLAIMED. IN NO EVENT SHALL THE
COPYRIGHT OWNER OR CONTRIBUTORS BE LIABLE FOR ANY DIRECT, INDIRECT,
INCIDENTAL, SPECIAL, EXEMPLARY, OR CONSEQUENTIAL DAMAGES (INCLUDING,
BUT NOT LIMITED TO, PROCUREMENT OF SUBSTITUTE GOODS OR SERVICES;
LOSS OF USE, DATA, OR PROFITS; OR BUSINESS INTERRUPTION) HOWEVER
CAUSED AND ON ANY THEORY OF LIABILITY, WHETHER IN CONTRACT, STRICT
LIABILITY, OR TORT (INCLUDING NEGLIGENCE OR OTHERWISE) ARISING IN
ANY WAY OUT OF THE USE OF THIS SOFTWARE, EVEN IF ADVISED OF THE
POSSIBILITY OF SUCH DAMAGE.
```

E.6 RFC 3174 - US Secure Hash Algorithm 1 (SHA1) License

The following software may be included in this product:

RFC 3174 - US Secure Hash Algorithm 1 (SHA1)

```
RFC 3174 - US Secure Hash Algorithm 1 (SHA1)
```

Acknowledgement
Funding for the RFC Editor function is currently provided by the
Internet Society.

MySQL Enterprise Backup Glossary

These terms are commonly used in information about the MySQL Enterprise Backup product.

A

.ARM file
: Metadata for ARCHIVE tables. Contrast with **.ARZ file**. Files with this extension are always included in backups produced by the `mysqlbackup` command of the **MySQL Enterprise Backup** product.
See Also .ARZ file, MySQL Enterprise Backup.

.ARZ file
: Data for ARCHIVE tables. Contrast with **.ARM file**. Files with this extension are always included in backups produced by the `mysqlbackup` command of the **MySQL Enterprise Backup** product.
See Also .ARM file, MySQL Enterprise Backup.

Antelope
: The code name for the original InnoDB **file format**. It supports the **redundant** and **compact** row formats, but not the newer **dynamic** and **compressed** row formats available in the **Barracuda** file format.

  If your application could benefit from InnoDB table **compression**, or uses BLOBs or large text columns that could benefit from the dynamic row format, you might switch some tables to Barracuda format. You select the file format to use by setting the `innodb_file_format` option before creating the table.
  See Also Barracuda, compression, file format.

apply
: The operation that transforms a **raw backup** into a **prepared backup** by incorporating changes that occurred while the backup was running, using data from the **log**.
See Also log, prepared backup, raw backup.

B

backup
: The process of copying some or all table data and metadata from a MySQL instance, for safekeeping. Can also refer to the set of copied files. This is a crucial task for DBAs. The reverse of this process is the **restore** operation.

  With MySQL, **physical backups** are performed by the **MySQL Enterprise Backup** product, and **logical backups** are performed by the `mysqldump` command. These techniques have different characteristics in terms of size and representation of the backup data, and speed (especially speed of the restore operation).

  Backups are further classified as **hot**, **warm**, or **cold** depending on how much they interfere with normal database operation. (Hot backups have the least interference, cold backups the most.)
  See Also cold backup, hot backup, logical backup, MySQL Enterprise Backup, mysqldump, physical backup, warm backup.

backup repository
: Contrast with **server repository**.
See Also repository, server repository.

backup-my.cnf
: A small **configuration file** generated by **MySQL Enterprise Backup**, containing a minimal set of configuration parameters. This file records the settings that apply to this backup data. Subsequent operations, such as the **apply** process, read options from this file to determine how the backup data is structured. This file always has the extension `.cnf`, rather than `.cnf` on Unix-like systems and `.ini` on Windows systems.
See Also apply, configuration file.

Barracuda
: The code name for an InnoDB **file format** that supports compression for table data. This file format was first introduced in the InnoDB Plugin. It supports the **compressed** row format that enables InnoDB table

compression, and the **dynamic** row format that improves the storage layout for BLOB and large text columns. You can select it through the `innodb_file_format` option.

Because the InnoDB **system tablespace** is stored in the original **Antelope** file format, to use the Barracuda file format you must also enable the **file-per-table** setting, which puts newly created tables in their own tablespaces separate from the system tablespace.

The **MySQL Enterprise Backup** product version 3.5 and above supports backing up tablespaces that use the Barracuda file format.
See Also Antelope, file format, MySQL Enterprise Backup, row format, system tablespace.

binary log
A file containing a record of all statements that attempt to change table data. These statements can be replayed to bring slave servers up to date in a **replication** scenario, or to bring a database up to date after restoring table data from a backup. The binary logging feature can be turned on and off, although Oracle recommends always enabling it if you use replication or perform backups.

You can examine the contents of the binary log, or replay those statements during replication or recovery, by using the `mysqlbinlog` command. For full information about the binary log, see The Binary Log. For MySQL configuration options related to the binary log, see Binary Log Options and Variables.

For the **MySQL Enterprise Backup** product, the file name of the binary log and the current position within the file are important details. To record this information for the master server when taking a backup in a replication context, you can specify the `--slave-info` option.

Prior to MySQL 5.0, a similar capability was available, known as the update log. In MySQL 5.0 and higher, the binary log replaces the update log.
See Also binlog, MySQL Enterprise Backup, replication.

binlog
An informal name for the **binary log** file. For example, you might see this abbreviation used in e-mail messages or forum discussions.
See Also binary log.

C

cold backup
A **backup** taken while the database is shut down. For busy applications and web sites, this might not be practical, and you might prefer a **warm backup** or a **hot backup**.
See Also backup, connection, hot backup, warm backup.

compression
A technique that produces smaller **backup** files, with size reduction influenced by the **compression level** setting. Suitable for keeping multiple sets of non-critical backup files. (For recent backups of critical data, you might leave the data uncompressed, to allow fast restore speed in case of emergency.)

MySQL Enterprise Backup can apply compression to the contents of **InnoDB** tables during the backup process, turning the **.ibd** files into **.ibz** files.

Compression adds CPU overhead to the backup process, and requires additional time and disk space during the **restore** process.
See Also backup, compression level, .ibd file, .ibz file, InnoDB, MySQL Enterprise Backup, restore.

compression level
A setting that determines how much **compression** to apply to a compressed backup. This setting ranges from 0 (none), 1 (default level when compression is enabled) to 9 (maximum). The amount of compression for a given compression level depends on the nature of your data values. Higher compression levels do impose additional CPU overhead, so ideally you use the lowest value that produces a good balance of compression with low CPU overhead.
See Also compression.

configuration file
: The file that holds the startup options of the MySQL server and related products and components. Often referred to by its default file name, **my.cnf** on Linux, Unix, and OS X systems, and **my.ini** on Windows systems. The **MySQL Enterprise Backup** stores its default configuration settings in this file, under a `[mysqlbackup]` section. For convenience, MySQL Enterprise Backup can also read settings from the `[client]` section, for configuration options that are common between MySQL Enterprise Backup and other programs that connect to the MySQL server.
See Also my.cnf, my.ini, MySQL Enterprise Backup.

connection
: The mechanism used by certain backup operations to communicate with a running MySQL **server**. For example, the `mysqlbackup` command can log into the server being backed up to insert and update data in the **progress table** and the **history table**. A **hot backup** typically uses a database connection for convenience, but can proceed anyway if the connection is not available. A **warm backup** always uses a database connection, because it must put the server into a read-only state. A **cold backup** is taken while the MySQL server is shut down, and so cannot use any features that require a connection.
See Also cold backup, history table, hot backup, progress table, server, warm backup.

crash recovery
: The cleanup activities for InnoDB tables that occur when MySQL is started again after a crash. Changes that were committed before the crash, but not yet written to the tablespace files, are reconstructed from the **doublewrite buffer**. When the database is shut down normally, this type of activity is performed during shutdown by the **purge** operation.

D

data dictionary
: A set of tables, controlled by the InnoDB storage engine, that keeps track of InnoDB-related objects such as tables, indexes, and table columns. These tables are part of the InnoDB **system tablespace**.

  Because the **MySQL Enterprise Backup** product always backs up the system tablespace, all backups include the contents of the data dictionary.
  See Also hot backup, MySQL Enterprise Backup, system tablespace.

database
: A set of tables and related objects owned by a MySQL user. Equivalent to “schema” in Oracle Database terminology. **MySQL Enterprise Backup** can perform a **partial backup** that includes some databases and not others. The full set of databases controlled by a MySQL server is known as an **instance**.
  See Also instance, MySQL Enterprise Backup, partial backup.

downtime
: A period when the database is unresponsive. The database might be entirely shut down, or in a read-only state when applications are attempting to insert, update, or delete data. The goal for your backup strategy is to minimize downtime, using techniques such as **hot backup** for **InnoDB** tables, **cold backup** using **slave** servers in a **replication** configuration, and minimizing the duration of the **suspend** stage where you run customized backup logic while the MySQL server is **locked**.
  See Also cold backup, hot backup, InnoDB, locking, replication, slave, suspend.

E

exclude
: In a **partial backup**, to select a set of tables, databases, or a combination of both to be omitted from the backup. Contrast with **include**.
  See Also partial backup.

extract
: The operation that retrieves some content from an **image** file produced by a **single-file backup**. It can apply to a single file (unpacked to an arbitrary location) or to the entire backup (reproducing the original directory

structure of the backup data). These two kinds of extraction are performed by the `mysqlbackup` options `extract` and `image-to-backup-dir`, respectively.
See Also image, single-file backup.

F

.frm file
A file containing the metadata, such as the table definition, of a MySQL table.

For backups, you must always keep the full set of `.frm` files along with the backup data to be able to restore tables that are altered or dropped after the backup.

Although each InnoDB table has a `.frm` file, InnoDB maintains its own table metadata in the system tablespace; the `.frm` files are not needed for InnoDB to operate on InnoDB tables.

These files are backed up by the **MySQL Enterprise Backup** product. These files must not be modified by an `ALTER TABLE` operation while the backup is taking place, which is why backups that include non-InnoDB tables perform a `FLUSH TABLES WITH READ LOCK` operation to freeze such activity while backing up the `.frm` files. Restoring a backup can result in `.frm` files being created, changed, or removed to match the state of the database at the time of the backup.
See Also MySQL Enterprise Backup.

file format
The format used by InnoDB for its data files named `ibdata1`, `ibdata2`, and so on. Each file format supports one or more row formats.
See Also Antelope, Barracuda, ibdata file, row format.

full backup
A **backup** that includes all the **tables** in each MySQL database, and all the databases in a MySQL instance. Contrast with **partial backup** and **incremental backup**. Full backups take the longest, but also require the least amount of followup work and administration complexity. Thus, even when you primarily do partial or incremental backups, you might periodically do a full backup.
See Also backup, incremental backup, partial backup, table.

H

history table
The table `mysql.backup_history` that holds details of completed **backup** operations. While a backup job is running, the details (especially the changing status value) are recorded in the **progress table**.
See Also backup, progress table.

hot backup
A backup taken while the MySQL **instance** and is running and applications are reading and writing to it. Contrast with **warm backup** and **cold backup**.

A hot backup involves more than simply copying data files: it must include any data that was inserted or updated while the backup was in process; it must exclude any data that was deleted while the backup was in process; and it must ignore any changes started by **transactions** but not committed.

The Oracle product that performs hot backups, of **InnoDB** tables especially but also tables from MyISAM and other storage engines, is **MySQL Enterprise Backup**.

The hot backup process consists of two stages. The initial copying of the InnoDB data files produces a **raw backup**. The **apply** step incorporates any changes to the database that happened while the backup was running. Applying the changes produces a **prepared** backup; these files are ready to be restored whenever necessary.

A **full backup** consists of a hot backup phase that copies the InnoDB data, followed by a **warm backup** phase that copies any non-InnoDB data such as MyISAM tables and **.frm** files.

See Also apply, cold backup, .frm file, full backup, InnoDB, instance, MySQL Enterprise Backup, prepared backup, raw backup, warm backup.

I

.ibd file
Each InnoDB **tablespace** created using the **file-per-table** setting has a filename with a `.ibd` extension. This extension does not apply to the **system tablespace**, which is made up of files named `ibdata1`, `ibdata2`, and so on.
See Also .ibz file, system tablespace, tablespace.

.ibz file
When the **MySQL Enterprise Backup** product performs a **compressed backup**, it transforms each **tablespace** file that is created using the **file-per-table** setting from a `.ibd` extension to a `.ibz` extension.

The compression applied during backup is distinct from the **compressed row format** that keeps table data compressed during normal operation. An InnoDB tablespace that is already in compressed row format is not compressed a second time, because that would save little or no space.
See Also .ibd file, .ibz file, MySQL Enterprise Backup, tablespace.

ibdata file
A set of files with names such as `ibdata1`, `ibdata2`, and so on, that make up the InnoDB **system tablespace**. These files contain metadata about InnoDB tables, and can contain some or all of the table and index data also (depending on whether the **file-per-table option** is in effect when each table is created). For backward compatibility these files always use the **Antelope** file format.
See Also Antelope, system tablespace.

image
The file produced as part of a **single-file backup** operation. It can be a real file that you store locally, or standard output (specified as `-`) when the backup data is **streamed** directly to another command or remote server. This term is referenced in several `mysqlbackup` options such as `backup-dir-to-image` and `image-to-backup-dir`.
See Also single-file backup, streaming.

include
In a **partial backup**, to select a set of tables, databases, or a combination of both to be backed up. Contrast with **exclude**.
See Also partial backup.

incremental backup
A backup that captures only data changed since the previous backup. It has the potential to be smaller and faster than a **full backup**. The incremental backup data must be merged with the contents of the previous backup before it can be restored. See Section 3.3.2, “Making an Incremental Backup” for usage details.
Related `mysqlbackup` options are `--incremental`, `--incremental-with-redo-log-only`, `--incremental-backup-dir`, `--incremental-base`, and `--start-lsn`.
See Also full backup.

InnoDB
The type of MySQL **table** that works best with **MySQL Enterprise Backup**. These tables can be backed up using the **hot backup** technique that avoids interruptions in database processing. For this reason, and because of the higher reliability and concurrency possible with InnoDB tables, most deployments should use InnoDB for the bulk of their data and their most important data. In MySQL 5.5 and higher, the `CREATE TABLE` statement creates InnoDB tables by default.
See Also hot backup, MySQL Enterprise Backup, table.

instance
The full contents of a MySQL server, possibly including multiple **databases**. A **backup** operation can back up an entire instance, or a **partial backup** can include selected databases and tables.

See Also database, partial backup.

L

locking

See Also suspend, warm backup.

log
Several types of log files are used within the MySQL Enterprise Backup product. The most common is the InnoDB **redo log** that is consulted during **incremental backups**.
See Also incremental backup, redo log.

log sequence number
See LSN.

logical backup
A **backup** that reproduces table structure and data, without copying the actual data files. For example, the `mysqldump` command produces a logical backup, because its output contains statements such as `CREATE TABLE` and `INSERT` that can re-create the data. Contrast with **physical backup**.
See Also backup, physical backup.

LSN
Acronym for **log sequence number**. This arbitrary, ever-increasing value represents a point in time corresponding to operations recorded in the **redo log**. (This point in time is regardless of transaction boundaries; it can fall in the middle of one or more transactions.) It is used internally by InnoDB during **crash recovery** and for managing the buffer pool.

In the **MySQL Enterprise Backup** product, you can specify an LSN to represent the point in time from which to take an **incremental backup**. The relevant LSN is displayed by the output of the `mysqlbackup` command. Once you have the LSN corresponding to the time of a full backup, you can specify that value to take a subsequent incremental backup, whose output contains another LSN for the next incremental backup.
See Also crash recovery, hot backup, incremental backup, redo log.

M

.MRG file
A file containing references to other tables, used by the `MERGE` storage engine. Files with this extension are always included in backups produced by the `mysqlbackup` command of the **MySQL Enterprise Backup** product.
See Also MySQL Enterprise Backup.

.MYD file
A file that MySQL uses to store data for a MyISAM table.
See Also .MYI file, MySQL Enterprise Backup.

.MYI file
A file that MySQL uses to store indexes for a MyISAM table.
See Also .MYD file, MySQL Enterprise Backup.

manifest
The record of the environment (for example, command-line arguments) and data files involved in a backup, stored in the files `meta/backup_create.xml` and `meta/backup_content.xml`, respectively. This data can be used by management tools during diagnosis and troubleshooting procedures.

master
In a **replication** configuration, a database server that sends updates to a set of **slave** servers. It typically dedicates most of its resources to write operations, leaving user queries to the slaves. With **MySQL**

Enterprise Backup, typically you perform backups on the slave servers rather than the master, to minimize any slowdown of the overall system.
See Also MySQL Enterprise Backup, replication, slave.

media management software
A class of software programs for managing backup media, such as libraries of tape backups. One example is **Oracle Secure Backup**. Abbreviated **MMS**.
See Also Oracle Secure Backup.

my.cnf
The typical name for the MySQL **configuration file** on Linux, Unix, and OS X systems.
See Also configuration file, my.ini.

my.ini
The typical name for the MySQL **configuration file** on Windows systems.
See Also configuration file, my.cnf.

MyISAM
A MySQL storage engine, formerly the default for new tables. In MySQL 5.5 and higher, **InnoDB** becomes the default storage engine. MySQL Enterprise Backup can back up both types of tables, and tables from other storage engines also. The backup process for InnoDB tables (**hot backup**) is less disruptive to database operations than for MyISAM tables (**warm backup**).
See Also hot backup, InnoDB, MySQL Enterprise Backup, warm backup.

MySQL Enterprise Backup
A licensed products that performs **hot backups** of MySQL databases. It offers the most efficiency and flexibility when backing up **InnoDB** tables; it can also back up MyISAM and other kinds of tables. It is included as part of the MySQL Enterprise Edition subscription.
See Also Barracuda, hot backup, InnoDB.

mysqlbackup
The primary command of the **MySQL Enterprise Backup** product. Different options perform **backup** and **restore** operations.
See Also backup, MySQL Enterprise Backup, restore.

mysqldump
A MySQL command that performs **logical backups**, producing a set of SQL commands to recreate tables and data. Suitable for smaller backups or less critical data, because the **restore** operation takes longer than with a **physical backup** produced by **MySQL Enterprise Backup**.
See Also logical backup, MySQL Enterprise Backup, physical backup, restore.

O

.opt file
A file containing database configuration information. Files with this extension are always included in backups produced by the backup operations of the **MySQL Enterprise Backup** product.
See Also MySQL Enterprise Backup.

offline
A type of operation performed while the database server is stopped. With the **MySQL Enterprise Backup** product, the main offline operation is the **restore** step. You can optionally perform a **cold backup**, which is another offline operation. Contrast with **online**.
See Also cold backup, MySQL Enterprise Backup, online, restore.

online
A type of operation performed while the database server is running. A **hot backup** is the ideal example, because the database continues to run and no read or write operations are blocked. For that reason, sometimes “hot backup” and “online backup” are used as synonyms. A **cold backup** is the opposite of

an online operation; by definition, the database server is shut down while the backup happens. A **warm backup** is also a kind of online operation, because the database server continues to run, although some write operations could be blocked while a warm backup is in progress. Contrast with **offline**.
See Also cold backup, hot backup, offline, warm backup.

Oracle Secure Backup
An Oracle product for managing **backup** media, and so classified as **media management software** (**MMS**). Abbreviated **OSB**. For **MySQL Enterprise Backup**, OSB is typically used to manage tape backups.
See Also backup, media management software, MySQL Enterprise Backup, OSB.

OSB
Abbreviation for **Oracle Secure Backup**, a **media management software** product (**MMS**).
See Also Oracle Secure Backup.

P

.par file
A file containing partition definitions. Files with this extension are always included in backups produced by the `mysqlbackup` command of the **MySQL Enterprise Backup** product.
See Also MySQL Enterprise Backup.

parallel backup
The default processing mode in MySQL Enterprise Backup 3.8 and higher, employing multiple threads for different classes of internal operations (read, process, and write). See Section 1.3, “Overview of Backup Performance and Capacity Considerations” for an overview, Section 4.1.11, “Performance / Scalability / Capacity Options” for the relevant `mysqlbackup` options, and Chapter 7, *Performance Considerations for MySQL Enterprise Backup* for performance guidelines and tips.

partial backup
A **backup** that contains some of the **tables** in a MySQL database, or some of the databases in a MySQL instance. Contrast with **full backup**.
See Also backup, full backup, partial restore, table.

partial restore
A **restore** operation that applies to one or more **tables** or **databases**, but not the entire contents of a MySQL server. The data being restored could come from either a **partial backup** or a **full backup**.
See Also database, full backup, partial backup, restore, table.

physical backup
A **backup** that copies the actual data files. For example, the **MySQL Enterprise Backup** command produces a physical backup, because its output contains data files that can be used directly by the `mysqld` server. Contrast with **logical backup**.
See Also backup, logical backup, MySQL Enterprise Backup.

point in time
The time corresponding to the end of a **backup** operation. A **prepared backup** includes all the changes that occurred while the backup operation was running. **Restoring** the backup brings the data back to the state at the moment when the backup operation completed.
See Also backup, prepared backup, restore.

prepared backup
The set of backup data that is entirely consistent and ready to be restored. It is produced by performing the **apply** operation on the **raw backup**.
See Also apply, raw backup.

progress table
The table `mysql.backup_progress` that holds details of running **backup** operations. When a backup job finishes, the details are recorded in the **history table**.

See Also backup, history table.

R

raw backup
: The initial set of backup data, not yet ready to be restored because it does not incorporate changes that occurred while the backup was running. The **apply** operation transforms the backup files into a **prepared backup** that is ready to be restored.
See Also apply, prepared backup.

redo log
: A set of files, typically named `ib_logfile0` and `ib_logfile1`, that record statements that attempt to change data in InnoDB tables. These statements are replayed automatically to correct data written by incomplete transactions, on startup following a crash. The passage of data through the redo logs is represented by the ever-increasing **LSN** value. The 4GB limit on maximum size for the redo log is raised in MySQL 5.6.
See Also LSN.

regular expression
: Some MySQL Enterprise Backup features use POSIX-style regular expressions, for example to specify tables, databases, or both to **include** or **exclude** from a **partial backup**. Regular expressions require escaping for dots in filenames, because the dot is the single-character wildcard; no escaping is needed for forward slashes in path names. When specifying regular expressions on the command line, surround them with quotation marks as appropriate for the shell environment, to prevent expansion of characters such as asterisks by the shell wildcard mechanism.
See Also exclude, include, partial backup.

replication
: A common configuration for MySQL deployments, with data and DML operations from a **master** server synchronized with a set of **slave** servers. With MySQL **Enterprise Backup,** you might take a backup on one server, and restore on a different system to create a new slave server with the data already in place. You might also back up data from a slave server rather than the master, to minimize any slowdown of the overall system.
See Also master, MySQL Enterprise Backup, slave.

repository
: We distinguish between the **server repository** and the **backup repository**.
See Also backup repository, server repository.

restore
: The converse of the **backup** operation. The data files from a **prepared backup** are put back into place to repair a data issue or bring the system back to an earlier state.
See Also backup, prepared backup.

row format
: The disk storage format for a row from an InnoDB table. As InnoDB gains new capabilities such as compression, new row formats are introduced to support the resulting improvements in storage efficiency and performance.

  Each table has its own row format, specified through the `ROW_FORMAT` option. To see the row format for each InnoDB table, issue the command `SHOW TABLE STATUS`. Because all the tables in the system tablespace share the same row format, to take advantage of other row formats typically requires setting the `innodb_file_per_table` option, so that each table is stored in a separate tablespace.

S

SBT
: Acronym for **system backup to tape**.

See Also system backup to tape.

server
A MySQL **instance** controlled by a `mysqld` daemon. A physical machine can host multiple MySQL servers, each requiring its own **backup** operations and schedule. Some backup operations communicate with the server through a **connection**.
See Also connection, instance.

server repository
Contrast with **backup repository**.
See Also backup repository, repository.

single-file backup
A backup technique that packs all the backup data into one file (the backup **image**), for ease of storage and transfer. The **streaming** backup technique requires using a single-file backup.
See Also image, streaming.

slave
In a **replication** configuration, a database server that receives updates from a **master** server. Typically used to service user queries, to minimize the query load on the master. With **MySQL Enterprise Backup**, you might take a backup on one server, and restore on a different system to create a new slave server with the data already in place. You might also back up data from a slave server rather than the master, to minimize any slowdown of the overall system.
See Also master, replication.

streaming
A backup technique that transfers the data immediately to another server, rather than saving a local copy. Uses mechanisms such as Unix pipes. Requires a **single-file backup**, with the destination file specified as `-` (standard output).
See Also single-file backup.

suspend
An optional stage within the backup where the MySQL Enterprise Backup processing stops, to allow for user-specific operations to be run. The `mysqlbackup` command has options that let you specify commands to be run while the backup is suspended. Most often used in conjunction with backups of **InnoDB** tables only, where you might do your own scripting for handling **.frm files**.
See Also .frm file, InnoDB.

system backup to tape
An API for **media management software**. Abbreviated **SBT**. Several `mysqlbackup` options (with **sbt** in their names) pass information to **media management software** products such as **Oracle Secure Backup**.
See Also Oracle Secure Backup, SBT.

system tablespace
By default, this single data file stores all the table data for a database, as well as all the metadata for InnoDB-related objects (the **data dictionary**).

Turning on the **innodb_file_per_table** option causes each newly created table to be stored in its own **tablespace**, reducing the size of, and dependencies on, the system tablespace.

Keeping all table data in the system tablespace has implications for the **MySQL Enterprise Backup** product (backing up one large file rather than several smaller files), and prevents you from using certain InnoDB features that require the newer **Barracuda** file format. on the
See Also Barracuda, data dictionary, file format, ibdata file, tablespace.

T

.TRG file
A file containing **trigger** parameters. Files with this extension are always included in backups produced by the `mysqlbackup` command of the **MySQL Enterprise Backup** product.

See Also MySQL Enterprise Backup.

table
: Although a table is a distinct, addressable object in the context of SQL, for **backup** purposes we are often concerned with whether the table is part of the **system tablespace**, or was created under the **file-per-table** setting and so resides in its own **tablespace**.
See Also backup, system tablespace, tablespace.

tablespace
: For **InnoDB** tables, the file that holds the data and indexes for a table. Can be either the **system tablespace** containing multiple tables, or a table created with the **file-per-table** setting that resides in its own tablespace file.
See Also InnoDB, system tablespace.

W

warm backup
: A **backup** taken while the database is running, but that restricts some database operations during the backup process. For example, tables might become read-only. For busy applications and web sites, you might prefer a **hot backup**.
See Also backup, cold backup, hot backup.

Index

Symbols

A

B

C

D

E

F

G

H

I

L

M

N

O