MySQL Enterprise Backup User's Guide (Version 3.12.5)

Abstract

This is the User's Guide for the MySQL Enterprise 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.

For notes detailing the changes in each release, see the MySQL Enterprise Backup 3.12 Release Notes.

For legal information, see the Legal Notices.

For help with using MySQL, please visit the MySQL Forums, where you can discuss your issues with other MySQL users.

Licensing information. This product may include third-party software, used under license. See the MySQL Enterprise Backup 3.12 License Information User Manual for licensing information, including licensing information relating to third-party software that may be included in this MySQL Enterprise Backup release.

Document generated on: 2023-03-03 (revision: 8501)

Table of Contents

Preface and Legal Notices ix
I Getting Started with MySQL Enterprise Backup 1
1 Introduction to MySQL Enterprise Backup 5
1.1 Types of Backups 5
1.2 The mysqlbackup Client 6
1.3 Overview of Backup Performance and Capacity Considerations 6
1.4 Files that Are Backed Up 7
1.5 Overview of Restoring a Database 20
2 Installing MySQL Enterprise Backup 21
3 What's New in MySQL Enterprise Backup 3.12? 23
II Using MySQL Enterprise Backup 25
4 Backing Up a Database Server 29
4.1 Before the First Backup 29
4.1.1 Collect Database Information 29
4.1.2 Grant MySQL Privileges to Backup Administrator 31
4.1.3 Designate a Location for Backup Data 32
4.2 The Typical Backup / Verify / Restore Cycle 32
4.2.1 OS User for Running mysqlbackup 32
4.2.2 Backing Up an Entire MySQL Instance 32
4.2.3 Verifying a Backup 34
4.2.4 Restoring a Database 35
4.3 Backup Scenarios and Examples 37
4.3.1 Making a Full Backup 37
4.3.2 Making a Differential or Incremental Backup 38
4.3.3 Making a Compressed Backup 42
4.3.4 Making a Partial Backup 43
4.3.5 Making a Single-File Backup 48
4.3.6 Making an Optimistic Backup 53
4.3.7 Making a Back Up of In-Memory Database Data 55
4.3.8 Making Scheduled Backups 55
4.4 Making Backups with a Distributed File System (DFS) or Storage Access Network (SAN) 56
5 Recovering or Restoring a Database 57
5.1 Preparing the Backup to be Restored 57
5.2 Performing a Restore Operation 58
5.2.1 Restoring a Compressed Backup 59
5.2.2 Restoring an Encrypted Backup Image 60
5.2.3 Restoring an Incremental Backup 60
5.2.4 Restoring Backups Created with the `--use-tts` Option 61
5.2.5 Restoring External InnoDB Tablespaces to Different Locations 62
5.2.6 Restoring a Backup from Cloud Storage to a MySQL Server 62
5.3 Point-in-Time Recovery 63
5.4 Restoring a Backup with a Database Upgrade or Downgrade 65
6 Using MySQL Enterprise Backup with Replication 67
6.1 Setting Up a New Replica 67
6.2 Backing up and Restoring a Replica Database 69
6.3 Restoring a Source Database 70
7 Performance Considerations for MySQL Enterprise Backup 73
7.1 Optimizing Backup Performance 73
7.2 Optimizing Restore Performance 76
8 Encryption for Backups 79

9 Using MySQL Enterprise Backup with Media Management Software (MMS) Products 81
9.1 Backing Up to Tape with Oracle Secure Backup 81
10 Monitoring Backups with MySQL Enterprise Monitor 83
11 Troubleshooting for MySQL Enterprise Backup 85
11.1 Error codes of MySQL Enterprise Backup 85
11.2 Working Around Corruption Problems 85
11.3 Using the MySQL Enterprise Backup Logs 86
11.4 Using the MySQL Enterprise Backup Manifest 88
III `mysqlbackup` Command Reference 89
12 `mysqlbackup` 93
13 `mysqlbackup` commands 95
13.1 Backup Operations 95
13.2 Apply-Log Operations 96
13.3 Restore Operations 97
13.4 Validation Operations 99
13.5 Single-File Backup Operations 101
14 `mysqlbackup` Command-Line Options 105
14.1 Standard Options 112
14.2 Connection Options 114
14.3 Server Repository Options 116
14.4 Backup Repository Options 118
14.5 Metadata Options 122
14.6 Compression Options 123
14.7 Incremental Backup Options 124
14.8 Partial Backup and Restore Options 127
14.9 Single-File Backup Options 133
14.10 Performance / Scalability / Capacity Options 136
14.11 Message Logging Options 143
14.12 Progress Report Options 144
14.13 Encryption Options 148
14.14 Cloud Storage Options 148
14.15 Options for Special Backup Types 151
15 Configuration Files and Parameters 155
IV Appendixes 157
A Frequently Asked Questions for MySQL Enterprise Backup 161
B Limitations of MySQL Enterprise Backup 163
C Compatibility Information for MySQL Enterprise Backup 165
C.1 Supported Platforms 165
C.2 Cross-Platform Compatibility 165
C.3 Compatibility with MySQL Versions 165
C.4 Compatibility with Older Versions of MySQL Enterprise Backup 165
C.5 Compatibility Notes for Specific MySQL Versions 165
D MySQL Enterprise Backup Release Notes 167
MySQL Enterprise Backup Glossary 169
Index 183

List of Tables

1.1 Files in a MySQL Enterprise Backup Output Directory 7
4.1 Information Needed to Back Up a Database 29
14.1 List of All Options 105

List of Examples

4.1 Making an Uncompressed Partial Backup of InnoDB Tables 47
4.2 Making a Compressed Partial Backup 47
4.3 Single-File Backup to Absolute Path 48
4.4 Single-File Backup to Relative Path 48
4.5 Single-File Backup to Standard Output 49
4.6 Convert Existing Backup Directory to Single Image 49
4.7 Extract Existing Image to Backup Directory 49
4.8 List Single-File Backup Contents 49
4.9 Validate a Single-File Backup 49
4.10 Extract Single-File Backup into Current Directory 49
4.11 Extract Single-File Backup into a Backup Directory 49
4.12 Selective Extract of Single File 49
4.13 Selective Extract of Single Directory 50
4.14 Dealing with Absolute Path Names 50
4.15 Single-File Backup to a Remote Host 50
4.16 Single-file Backup to a Remote MySQL Server 51
4.17 Stream a Backup Directory to a Remote MySQL Server 51
4.18 Creating a Cloud Backup on Oracle Cloud Infrastructure Object Storage Classic 52
4.19 Creating a Cloud Backup on Oracle Cloud Infrastructure Object Storage 52
4.20 Creating a Cloud Incremental Backup on Oracle Cloud Infrastructure Object Storage 52
4.21 Creating a Cloud Backup on an OpenStack Object Storage 52
4.22 Creating a Cloud Backup on Amazon S3 53
4.23 Extract an Existing Image from an Oracle Cloud Infrastructure Object Storage Classic Container to a Backup Directory 53
4.24 Extract an Existing Image from Amazon S3 Cloud Storage to a Backup Directory 53
4.25 Optimistic Backup Using the Option `optimistic-time=YYMMDDHHMMSS` 54
4.26 Optimistic Backup Using the Option `optimistic-time=now` 54
4.27 Optimistic Backup Using the `optimistic-busy-tables` Option 54
4.28 Optimistic and Partial Backup Using both the `optimistic-busy-tables` and `optimistic-time` Options 55
5.1 Applying the Log to a Backup 58
5.2 Applying the Log to a Compressed Backup 58
5.3 Applying an Incremental Backup to a Full Backup 58
5.4 Shutting Down and Restoring a Database 58
5.5 Restoring a Backup Directory using `copy-back-and-apply-log` 59
5.6 Restoring a Single-file Backup using `copy-back-and-apply-log` 59
5.7 Restoring a Compressed Backup 59
5.8 Restoring an Encrypted Backup Image 60
5.9 Restoring an Incremental Backup Image 60
5.10 Restoring Selected Tables from a TTS Backup 61
5.11 Restoring and Renaming a Table from a TTS Backup 62
5.12 Restoring a Single-file Backup from an Oracle Cloud Infrastructure (OCI) Object Storage Classic Container to a MySQL Server 62
5.13 Restoring a Single-file Backup from an Oracle Cloud Infrastructure (OCI) Object Storage to a MySQL Server 62
5.14 Restoring a Cloud Incremental Backup from an Oracle Cloud Infrastructure (OCI) Object Storage Service to a MySQL Server 62
5.15 Restoring a Single-file Backup from an OpenStack Object Storage to a MySQL Server 63
5.16 Restoring a Single-file Backup from Amazon S3 to a MySQL Server 63
9.1 Sample `mysqlbackup` Commands Using MySQL Enterprise Backup with Oracle Secure Backup ... 82
13.1 Apply Log to Full Backup 97

15.1 Example `backup-my.cnf` file 155

Preface and Legal Notices

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

Licensing information. This product may include third-party software, used under license. See the MySQL Enterprise Backup 3.12 License Information User Manual for licensing information, including licensing information relating to third-party software that may be included in this MySQL Enterprise Backup release.

Legal Notices

Copyright © 2003, 2023, Oracle and/or its affiliates.

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

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

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

U.S. GOVERNMENT END USERS: Oracle programs (including any operating system, integrated software, any programs embedded, installed or activated on delivered hardware, and modifications of such programs) and Oracle computer documentation or other Oracle data delivered to or accessed by U.S. Government end users are "commercial computer software" or "commercial computer software documentation" pursuant to the applicable Federal Acquisition Regulation and agency-specific supplemental regulations. As such, the use, reproduction, duplication, release, display, disclosure, modification, preparation of derivative works, and/or adaptation of i) Oracle programs (including any operating system, integrated software, any programs embedded, installed or activated on delivered hardware, and modifications of such programs), ii) Oracle computer documentation and/or iii) other Oracle data, is subject to the rights and limitations specified in the license contained in the applicable contract. The terms governing the U.S. Government's use of Oracle cloud services are defined by the applicable contract for such services. No other rights are granted to the U.S. Government.

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

Oracle, Java, and MySQL are registered trademarks of Oracle and/or its affiliates. Other names may be trademarks of their respective owners.

Intel and Intel Inside are trademarks or registered trademarks of Intel Corporation. All SPARC trademarks are used under license and are trademarks or registered trademarks of SPARC International, Inc. AMD, Epyc, and the AMD logo are trademarks or registered trademarks of Advanced Micro Devices. UNIX is a registered trademark of The Open Group.

This software or hardware and documentation may provide access to or information about content, products, and services from third parties. Oracle Corporation and its affiliates are not responsible for and

expressly disclaim all warranties of any kind with respect to third-party content, products, and services unless otherwise set forth in an applicable agreement between you and Oracle. Oracle Corporation and its affiliates will not be responsible for any loss, costs, or damages incurred due to your access to or use of third-party content, products, or services, except as set forth in an applicable agreement between you and Oracle.

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

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

Documentation Accessibility

For information about Oracle's commitment to accessibility, visit the Oracle Accessibility Program website at
https://www.oracle.com/corporate/accessibility/.

Access to Oracle Support for Accessibility

Oracle customers that have purchased support have access to electronic support through My Oracle Support. For information, visit
https://www.oracle.com/corporate/accessibility/learning-support.html#support-tab.

Part I Getting Started with MySQL Enterprise Backup

Table of Contents

1 Introduction to MySQL Enterprise Backup 5
1.1 Types of Backups 5
1.2 The mysqlbackup Client 6
1.3 Overview of Backup Performance and Capacity Considerations 6
1.4 Files that Are Backed Up 7
1.5 Overview of Restoring a Database 20
2 Installing MySQL Enterprise Backup 21
3 What's New in MySQL Enterprise Backup 3.12? 23

Chapter 1 Introduction to MySQL Enterprise Backup

Table of Contents

1.1 Types of Backups 5
1.2 The mysqlbackup Client 6
1.3 Overview of Backup Performance and Capacity Considerations 6
1.4 Files that Are Backed Up 7
1.5 Overview of Restoring a Database 20

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 replica, 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 Client

When using the MySQL Enterprise Backup product, you primarily work with the `mysqlbackup` client. Use it to perform different types of backup and restore operations, as well as tasks that are otherwise performed by 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 information about the various `mysqlbackup` commands and command-line options, see Part III, “`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 4.3.5.1, “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 (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.

File Name, Pattern, or Extension	Relation to Original Data Files	Notes
`*.ibd`	InnoDB file-per-table tablespaces, each containing a single InnoDB table and associated indexes.	Used for tables created using the `innodb_file_per_table` option. 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.
`*.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 during the `apply-log`, `copy-back`, or `copy-back-and-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.

File Name, Pattern, or Extension	Relation to Original Data Files	Notes
`*.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.
`*.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 table 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 table 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 and other important information about the MySQL data files.	The file is created during a backup, and it contains crucial parameters describing the backed-up data like `innodb_data_file_path`, `innodb_log_file_size`, `innodb_log_files_in_group`, and so on. It might also contain other InnoDB parameters like `innodb_data_home_dir` and `innodb_undo_directory` if some of the backup repository options were used during the backup. `mysqlbackup` uses the parameters stored in this file to understand the structure of the backup and to perform various operations. You might need to supply some of these parameters to `mysqlbackup` during a restore and to `mysqld` when you start the target server if the target server and the backup are configured differently. See the discussion in Section 4.2.4, “Restoring a Database” for details.

File Name, Pattern, or Extension	Relation to Original Data Files	Notes
`ibbackup_ibd_files`	Records names of the `.ibd` files and their space IDs during an incremental backup.	This file is created during an incremental backup. During a restore, the information in the file is used to delete the tables from the full backup that has been removed between the time of the full backup and the time of the incremental backup.
`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 the database's 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`	Created instead of the `ibbackup_logfile` for incremental backups taken with the `--incremental-with-redo-log-only` option.	
`ib_logfile*`	Created in the backup directory by `mysqlbackup` 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 Creating Tables Externally 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 server where the backup is restored, `mysqlbackup` attempts to create it. If the directory cannot be created, the restore operation fails. Thus, if you would like to use the `DATA DIRECTORY` clause to

File Name, Pattern, or Extension	Relation to Original Data Files	Notes
		put tables at different locations or to 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. If the directory on the target server pointed to by a `.bl` file already contains .ibd files, the `--force` option is required when you restore the backup.
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 to easily keep more than one set of backup data under the same main backup directory.
`datadir` directory	A subdirectory that stores the data files and database subdirectories from the original MySQL instance.	Created under the backup directory by `mysqlbackup`.
binary log files	Binary log files from the server, which are included in a backup by default (except when the backup is created with the `--use-tts` option). They allow a snapshot of the server to be taken, so a server can be cloned to its exact state. Using a full backup as a basis, the binary log files that are included with an incremental backup can be used for a point-in-time recovery (PITR), which restores a database to its state at a certain point in time after the last full backup. See Section 5.3, “Point-in-Time Recovery” for details.	Saved under the `datadir` directory under the backup directory. Use the `--skip-binlog` option to exclude binary logs in the backup. For MySQL 5.5 and earlier, as well as all offline backups, use the `--log-bin-index` option to specify the absolute path of the index file on the MySQL server that lists all the used binary log files, if it is different from the default value of the option, for `mysqlbackup` to find the binary log files and include them in the backups. The index file itself, with the locations of the binary log files properly updated to point to the files' locations in the backup directory, is included into the backup under the `datadir` directory. The binary log files are compressed and saved with the `.bz` extension when being included in a compressed backup. The binary log files and the index file, when included in a backup, are always copied into the

File Name, Pattern, or Extension	Relation to Original Data Files	Notes
		restored server's data directory during a restore operation. Use the `--skip-binlog` option to skip the restoring of the binary log. **Notes** • If any binary log files are missing on the server you are backing up, you should use the `--skip-binlog` option to avoid `mysqlbackup` throwing an error for the missing files. • No binary log files are copied into the incremental backup if the `--use-tts` option or the `--start-lsn` option is used. To include binary log files for the period

File Name, Pattern, or Extension	Relation to Original Data Files	Notes
		covered by the incremental backup, do not use the `--use-tts` option and, instead of `--start-lsn`, use the `--incremental-base` option, which provides the necessary information for `mysqlbackup` to ensure that no gap exists between binary log data included in the previous backup and the current incremental backup.
relay log files	Relay log files from a replica server, which are included in a backup of a replica server by default (except when the backup is created with the `--use-tts` option). Their inclusion saves the time and resources required for fetching the relay logs from the source when the replica is being restored.	Saved under the `datadir` directory under the backup directory. Use the `--skip-relaylog` option to exclude relay logs in the backup. For offline backup, use the `--relay-log-index` option to specify the absolute path of the index file on the MySQL server that lists all the used relay log files, if it is different from the default value of

File Name, Pattern, or Extension	Relation to Original Data Files	Notes
		the option, for `mysqlbackup` to find the relay log files and include them in the backups. The index file itself, with the locations of the relay log files properly updated to point to the files' locations in the backup directory, is included into the backup under the `datadir` directory. The relay log files are compressed and saved with the `.bz` extension when being included in a compressed backup. The relay log files and the index file, when included in a backup, are always copied into the restored server's data directory during a restore operation. Use the `--skip-relaylog` option to skip the restoring of the relay log.
*.bz	Compressed binary log or relay log files.	The binary log and relay log files are compressed and saved with the `.bz` extension when being included in a compressed backup. They are decompressed during a restore.
replication metadata repository files	Usually named `master.info` and `relay-log.info`, they are included by default in a backup of a replica database in a replication setup. See Replication Metadata Repositories, for details.	Saved under the `datadir` directory under the backup directory. For an offline backup, use the`--master-info-file` and `--relaylog-info-file` options to specify the absolute paths of the information files, if they are different from the default values of the options, for `mysqlbackup` to find those files and include them in the backups. The copying of these files are skipped during a backup or a restore when the `--skip-relay-log` option is used.
Backup image file	A single-file backup produced by the `backup-to-image` command, 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,

File Name, Pattern, or Extension	Relation to Original Data Files	Notes
		then unpack it with `mysqlbackup` 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 in subdirectories under the `datadir` directory (that is, under *`backup-dir/datadir/subdir`*)	Copied from the database subdirectories under the MySQL data directory.	By default, any unrecognized files in subdirectories under the MySQL data directory are copied to the backup. To omit such files, specify the `--only-known-file-types` option. **Note** Some limitations apply to this behavior. See the discussion here in Appendix B, *Limitations of MySQL Enterprise Backup*.
`meta` directory	A subdirectory that stores files with metadata about the backup.	Created under the backup directory by `mysqlbackup`. All files listed below go inside the `meta` subdirectory.
`backup_variables.txt`	Holds important information about the backup. For use by `mysqlbackup` only.	`mysqlbackup` 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 11.4, “Using the MySQL Enterprise Backup Manifest”.	This file is not modified at any stage once generated.

File Name, Pattern, or Extension	Relation to Original Data Files	Notes
`backup_create.xml`	Lists the command line arguments and environment in which the backup was created. For details about this file, see Section 11.4, "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. It also contains details of all the plugins defined on the backed-up server, by which users should make sure the same plugins are defined in the same manner on the target server for restoration. For details about this file, see Section 11.4, "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.
`backup_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 using `mysqlbackup`, the file named `backup_gtid_executed.sql` is created in the `meta` folder under the backup directory. Edit and execute this file after restoring the backup data on a replica server; see Section 6.1, "Setting Up a New Replica" for details.
`server-my.cnf`	Contains values of the backed-up server's global variables that are set to non-default values. Use this file or `server-all.cnf` to start the target server for restoration.	During a `copy-back` or `copy-back-and-apply-log` operation, the server repository options values (e.g., `--datadir`, `--innodb_data_home_dir`, etc.) in the file are modified if the command makes changes to them through the command options. However, during an `apply-incremental-backup` operation, the values already saved in the file take precedence and they are not modified by the

File Name, Pattern, or Extension	Relation to Original Data Files	Notes
		option values supplied through the command. **Warning** When using the file to restart the target server, change parameters like `--tmpdir`, `--general-log`, etc., and any global variable that uses an absolute filepath to avoid the accidental usage of the wrong file locations by the target server.
`server-all.cnf`	Contains values of all the global variables of the backed-up server. Use this file or `server-my.cnf` to start the target server for restoration.	During a `copy-back` or `copy-back-and-apply-log` operation, the server repository options values (e.g., `--datadir`, `--innodb_data_home_dir`, etc.) in the file are modified if the command makes changes to them through the command options. However, during an `apply-incremental-backup` operation, the values already saved in the file take precedence and they are not modified by the option values supplied through the command.

File Name, Pattern, or Extension	Relation to Original Data Files	Notes
		Warning When using the file to restart the target server, change parameters like `--tmpdir`, `--general-log`, etc., and any global variable that uses an absolute filepath to avoid the accidental usage of the wrong file locations by the target 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 (which represent the system tablespace and possibly the data for some user tables), any .ibd files (which contains data from user tables created with the file-per-table setting enabled), and the 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 replica.

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, `mysqlbackup` automatically copies the `.frm` files during a backup operation and restores the files during a restore operation.

Data from MyISAM and Other Storage Engines

`mysqlbackup` also backs up the .MYD files, .MYI files, and the .frm files associated with the MyISAM tables. Files with other extensions that are backed up are 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. They are backed up using the warm backup technique: 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:

- `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.
- `backup-my.cnf`: Records the crucial configuration parameters that apply to the backup. These configuration parameters are read by `mysqlbackup` during operations like `apply-log` to determine how the backup data is structured. These parameters are also checked during a restore operation for their compatibility with your target server's configuration.
- `server-my.cnf`: Contains values of the backed-up server's global variables that are set to non-default values.
- `server-all.cnf`: Contains values of all the global variables of the backed-up server.

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 want to perform a single-file backup rather than a directory backup, which produces a separate file for every file on the original server instance. A single-file backup is easier to transfer to a different system, to compress, and to uncompress; it also helps to prevent the situation in which individual files that form parts of a backup are deleted 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 4.3.5, “Making a Single-File Backup”.

1.5 Overview of Restoring a Database

To initiate the restore process, you run the `mysqlbackup` client with the `copy-back` or the `copy-back-and-apply-log` command. You can restore all the data for a MySQL server with multiple databases, each containing multiple tables. For backups created using transportable tablespace (TTS) (that is, backups created with the `--use-tts` option), you can also choose to 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 replica with the data from a source server, or to clone a database for reporting purposes.

See Chapter 5, *Recovering or Restoring a Database* for instructions on restoring databases.

Chapter 2 Installing MySQL Enterprise Backup

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

Optional: You can also install MySQL Enterprise Backup on computers other than the database server, only to run `mysqlbackup` with the `apply-log` option. See Section 13.2, “Apply-Log Operations” for information about bringing backup data to a separate server and running the “apply log” step there.

MySQL Enterprise Backup is packaged as either an archive file (`.tgz`, archived with `tar` and compressed with `gzip`) or as a platform-specific installer.

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
```

`mysqlbackup` 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` client is installed in the directory `/opt/mysql/meb-3.12`. You must add this directory to your `$PATH` setting.

Installing on Windows Systems

The product can be installed together with other MySQL products with the MySQL Installer for Windows. It can also be installed separately with either an individual `.msi` installer or `.zip` file.

When installing with a `.msi` installer, specify the installation location, preferably under the same directory where other MySQL products have been installed. Choose the option **Include directory in Windows PATH**, so that you can run `mysqlbackup` from any directory.

When installing with a `.zip` file, simply unzip the file and put `mysqlbackup.exe` at the desired installation location. You can add that location to the `%PATH%` variable, so that you can run the `mysqlbackup` client from any directory.

Verify the installation by selecting the menu item **Start > Programs > MySQL Enterprise Backup 3.12 > MySQL Enterprise Backup Command Line**. The menu item displays version information and opens a command prompt for running the `mysqlbackup` command.

Chapter 3 What's New in MySQL Enterprise Backup 3.12?

This chapter highlights the new features in MySQL Enterprise Backup 3.12, as well as any significant changes made to MySQL Enterprise Backup with the release of this series.

- **Renaming a table restored from a TTS backup.** You can now rename a table when you restore it from a backup created using transportable tablespace (TTS). See the description for the `--rename` option for details.
- **Support for OpenStack Object Storage.** MySQL Enterprise Backup now supports cloud backup and restore using OpenStack Object Storage ("Swift"). MySQL Enterprise Backup supports the Swift v1.0 API, and also the OpenStack Identity (Keystone) API v2.0 for authentication. Also supports authentication using Swift's TempAuth system. A number of new command options have been introduced to support OpenStack Object Storage; see Section 14.14, "Cloud Storage Options", for details.
- **Binary and Relay Logs are now compressed when included in a compressed backup.** The binary log file and relay log files (in the case of a replica server) are now compressed when they are being included in a compressed backup and decompressed during a restore.
- **Copying of the binary and relay Logs can now be skipped during a restore.** The `--skip-binlog` and `--skip-relaylog` options can now be used to skip the copying back of the binary log and relay log onto a server during a restore. This is particularly useful for users who do not want those logs appearing in the restored server's data directory, as that is always the location to which `mysqlbackup` will restore them, regardless of their original locations on the backed-up server.

Part II Using MySQL Enterprise Backup

Table of Contents

4 Backing Up a Database Server 29
4.1 Before the First Backup 29
4.1.1 Collect Database Information 29
4.1.2 Grant MySQL Privileges to Backup Administrator 31
4.1.3 Designate a Location for Backup Data 32
4.2 The Typical Backup / Verify / Restore Cycle 32
4.2.1 OS User for Running mysqlbackup 32
4.2.2 Backing Up an Entire MySQL Instance 32
4.2.3 Verifying a Backup 34
4.2.4 Restoring a Database 35
4.3 Backup Scenarios and Examples 37
4.3.1 Making a Full Backup 37
4.3.2 Making a Differential or Incremental Backup 38
4.3.3 Making a Compressed Backup 42
4.3.4 Making a Partial Backup 43
4.3.5 Making a Single-File Backup 48
4.3.6 Making an Optimistic Backup 53
4.3.7 Making a Back Up of In-Memory Database Data 55
4.3.8 Making Scheduled Backups 55
4.4 Making Backups with a Distributed File System (DFS) or Storage Access Network (SAN) 56
5 Recovering or Restoring a Database 57
5.1 Preparing the Backup to be Restored 57
5.2 Performing a Restore Operation 58
5.2.1 Restoring a Compressed Backup 59
5.2.2 Restoring an Encrypted Backup Image 60
5.2.3 Restoring an Incremental Backup 60
5.2.4 Restoring Backups Created with the `--use-tts` Option 61
5.2.5 Restoring External InnoDB Tablespaces to Different Locations 62
5.2.6 Restoring a Backup from Cloud Storage to a MySQL Server 62
5.3 Point-in-Time Recovery 63
5.4 Restoring a Backup with a Database Upgrade or Downgrade 65
6 Using MySQL Enterprise Backup with Replication 67
6.1 Setting Up a New Replica 67
6.2 Backing up and Restoring a Replica Database 69
6.3 Restoring a Source Database 70
7 Performance Considerations for MySQL Enterprise Backup 73
7.1 Optimizing Backup Performance 73
7.2 Optimizing Restore Performance 76
8 Encryption for Backups 79
9 Using MySQL Enterprise Backup with Media Management Software (MMS) Products 81
9.1 Backing Up to Tape with Oracle Secure Backup 81
10 Monitoring Backups with MySQL Enterprise Monitor 83
11 Troubleshooting for MySQL Enterprise Backup 85
11.1 Error codes of MySQL Enterprise Backup 85
11.2 Working Around Corruption Problems 85
11.3 Using the MySQL Enterprise Backup Logs 86
11.4 Using the MySQL Enterprise Backup Manifest 88

Chapter 4 Backing Up a Database Server

Table of Contents

4.1 Before the First Backup 29
4.1.1 Collect Database Information 29
4.1.2 Grant MySQL Privileges to Backup Administrator 31
4.1.3 Designate a Location for Backup Data 32
4.2 The Typical Backup / Verify / Restore Cycle 32
4.2.1 OS User for Running mysqlbackup 32
4.2.2 Backing Up an Entire MySQL Instance 32
4.2.3 Verifying a Backup 34
4.2.4 Restoring a Database 35
4.3 Backup Scenarios and Examples 37
4.3.1 Making a Full Backup 37
4.3.2 Making a Differential or Incremental Backup 38
4.3.3 Making a Compressed Backup 42
4.3.4 Making a Partial Backup 43
4.3.5 Making a Single-File Backup 48
4.3.6 Making an Optimistic Backup 53
4.3.7 Making a Back Up of In-Memory Database Data 55
4.3.8 Making Scheduled Backups 55
4.4 Making Backups with a Distributed File System (DFS) or Storage Access Network (SAN) 56

This section explains the preparations you need for creating backups with MySQL Enterprise Backup, the typical backup-verify-restore cycle, and the different backup scenarios for using MySQL Enterprise Backup. It also includes sample commands and outputs, showing you how to use the `mysqlbackup` client in different situations.

4.1 Before the First Backup

This section outlines some of the preparations needed before you can start working with MySQL Enterprise Backup.

4.1.1 Collect Database Information

Before backing up a particular database server for the first time, gather some information and use it to make some planning decisions, as outlined in the following table.

Table 4.1 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 the `--defaults-file` option in the `mysqld` startup script.	This is the preferred way to convey database configuration information to `mysqlbackup`, 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.

Information to Gather	Where to Find It	How Used
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 a cold (offline) backup, which works directly 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, and 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 at the terminal if the `--password` option is present without the password argument.
Path under which to store backup data	You choose this. See Section 4.1.3, “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 perform the backup and restore using a different OS user than the one who runs the server, this information might become important. See Section 4.2.1, “OS User for Running mysqlbackup” for details.
Size of InnoDB redo log files	Calculated from the values of the `innodb_log_file_size` and	Only needed if you perform incremental backups using the

Information to Gather	Where to Find It	How Used
	`innodb_log_files_in_group` configuration variables. Use the technique explained for the `--incremental-with-redo-log-only` option.	`--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 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.

4.1.2 Grant MySQL Privileges to Backup Administrator

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

The minimum privileges for the MySQL user with which `mysqlbackup` connects to the server are:

- `RELOAD` on all databases and tables.
- `CREATE`, `INSERT`, `DROP`, and `UPDATE` on the tables `mysql.backup_progress` and `mysql.backup_history`, and also `SELECT` and `ALTER` on `mysql.backup_history`.
- `SUPER`, to enable and disable logging, and to optimize locking in order to minimize disruption to database processing.
- `REPLICATION CLIENT`, to retrieve the binary log position, which is stored with the backup.

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

```
GRANT RELOAD ON *.* TO 'mysqlbackup'@'localhost';
GRANT CREATE, INSERT, DROP, UPDATE ON mysql.backup_progress TO 'mysqlbackup'@'localhost';
GRANT CREATE, INSERT, SELECT, ALTER, DROP, UPDATE ON mysql.backup_history
    TO 'mysqlbackup'@'localhost';
GRANT REPLICATION CLIENT ON *.* TO 'mysqlbackup'@'localhost';
GRANT SUPER ON *.* TO 'mysqlbackup'@'localhost';
```

The following additional privileges are required for using transportable tablespaces (TTS) to back up and restore InnoDB tables:

- `LOCK TABLES` and `SELECT` for backing up tables.
- `CREATE` for restoring tables.
- `DROP` for dropping tables if the restore fails for some reasons.
- `FILE` for restoring tables in external tablespaces outside of the server's data director.

To set these privileges, issue a statement like the following from the `mysql` client program:

```
GRANT LOCK TABLES, SELECT, CREATE, DROP, FILE ON *.* TO 'mysqlbackup'@'localhost';
```

4.1.3 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, referred to as the backup directory in this manual. Choose in advance for this directory a location on a file system with sufficient storage, which could even be remotely mounted from a different server. You specify the path to this directory with the `--backup-dir` option for many `mysqlbackup` commands.

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 `mysqlbackup` 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 replica), you might specify a new directory each time, or, for a single-file backup, specify the `--force` option to overwrite the old backup file.

4.2 The Typical Backup / Verify / Restore Cycle

To illustrate the basic steps in creating and making use of a backup, the following example shows how to perform a full backup, verify it, and then restore it to a server.

4.2.1 OS User for Running mysqlbackup

For Linux and other Unix-like platforms: `mysqlbackup` does not record file ownership or permissions of the files that are backed up. To ensure no file permission issues prevent a server to be backed up, restored, and restarted successfully, *it is highly recommended that you run mysqlbackup with the same OS user who runs the MySQL server (typically `mysql`)*. If that is not possible, pay attention to the following guidelines:

- For backups, `mysqlbackup` should be run by a user that can read all the server files and directories and can execute all server directories. To satisfy that requirement, the OS user that runs `mysqlbackup` should, for example, have the group owner of the server files and directories (typically `mysql`) as its primary group or secondary group.
- For restores, unless `mysqlbackup` is run by the same user that runs the server, it can be very hard to ensure that the server has access to all the restored server files and folders, especially in the case of an online restore, where the server must be able to access the files immediately after they are restored. For an offline restore, you might need to, for example, set a `umask` to the user before the restore and adjust the permissions of the restored files and folders using a series of `chmod` and `chown` commands, so that the original permissions for the backed-up files and folders are reproduced.

4.2.2 Backing Up an Entire MySQL Instance

In the following example, we back up an entire MySQL instance to a single file using the `backup-to-image` command, which appears at the end of the sample command. We specify some of the connection information for the database using the `--user` and `--host` options (and, with the `--password` option, tell the MySQL server to prompt for a user password). The location and filename for the single-file backup is specified using the `--backup-image` option, and the location for an empty folder to store temporary files is supplied with the `--backup-dir` option.

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 MySQL instance, allowing you to monitor long-running backups and view information on 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 has just been made.

```
$ ./mysqlbackup --user=root --password --host=127.0.0.1 --backup-image=/home/admin/backups/my.mbi \
    --backup-dir=/home/admin/backup-tmp backup-to-image

MySQL Enterprise Backup version 3.12.5 Linux-2.6.18-274.el5-i686 [2014/11/12]
Copyright (c) 2003, 2014, Oracle and/or its affiliates. All Rights Reserved.

 mysqlbackup: INFO: Starting with following command line ...
 ./mysqlbackup --user=root --password --host=127.0.0.1
        --backup-image=/home/admin/backups/my.mbi
        --backup-dir=/home/admin/backup-tmp backup-to-image

 mysqlbackup: INFO:
Enter password:
 mysqlbackup: INFO: MySQL server version is '5.6.17-log'.
 mysqlbackup: 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!".

141204 12:54:55 mysqlbackup: INFO: MEB logfile created at /home/admin/backup-tmp/meta/MEB_2014-12-04.12-54

--------------------------------------------------------------------
                      Server Repository Options:
--------------------------------------------------------------------
  datadir = /var/lib/mysql/
  innodb_data_home_dir =
  innodb_data_file_path = ibdata1:12M:autoextend
  innodb_log_group_home_dir = /var/lib/mysql/
  innodb_log_files_in_group = 2
  innodb_log_file_size = 50331648
  innodb_page_size = 16384
  innodb_checksum_algorithm = innodb
  innodb_undo_directory = /var/lib/mysql/
  innodb_undo_tablespaces = 0
  innodb_undo_logs = 128

--------------------------------------------------------------------
                      Backup Config Options:
--------------------------------------------------------------------
  datadir = /home/admin/backup-tmp/datadir
  innodb_data_home_dir = /home/admin/backup-tmp/datadir
  innodb_data_file_path = ibdata1:12M:autoextend
  innodb_log_group_home_dir = /home/admin/backup-tmp/datadir
  innodb_log_files_in_group = 2
  innodb_log_file_size = 50331648
  innodb_page_size = 16384
  innodb_checksum_algorithm = innodb
  innodb_undo_directory = /home/admin/backup-tmp/datadir
  innodb_undo_tablespaces = 0
  innodb_undo_logs = 128

Backup Image Path = /home/admin/backups/my.mbi
 mysqlbackup: INFO: Unique generated backup id for this is 14177156956623554

 mysqlbackup: INFO: Creating 14 buffers each of size 16777216.
141204 12:54:58 mysqlbackup: INFO: Full Image Backup operation starts with following threads
  1 read-threads    6 process-threads    1 write-threads
```

```
141204 12:54:58 mysqlbackup: INFO: System tablespace file format is Antelope.
141204 12:54:58 mysqlbackup: INFO: Starting to copy all innodb files...
 mysqlbackup: INFO: Copying meta file /home/admin/backup-tmp/backup-my.cnf.
 mysqlbackup: INFO: Copying meta file /home/admin/backup-tmp/meta/backup_create.xml.
141204 12:54:58 mysqlbackup: INFO: Found checkpoint at lsn 1631766.
141204 12:54:58 mysqlbackup: INFO: Starting log scan from lsn 1631744.
141204 12:54:58 mysqlbackup: INFO: Copying log...
141204 12:54:58 mysqlbackup: INFO: Log copied, lsn 1631766.
141204 12:54:58 mysqlbackup: INFO: Copying /var/lib/mysql/ibdata1 (Antelope file format).
141204 12:54:59 mysqlbackup: INFO: Copying /var/lib/mysql/mysql/innodb_index_stats.ibd (Antelope file format).
141204 12:54:59 mysqlbackup: INFO: Copying /var/lib/mysql/mysql/innodb_table_stats.ibd (Antelope file format).
141204 12:54:59 mysqlbackup: INFO: Copying /var/lib/mysql/mysql/slave_master_info.ibd (Antelope file format).
141204 12:55:00 mysqlbackup: INFO: Copying /var/lib/mysql/mysql/slave_relay_log_info.ibd (Antelope file format
141204 12:55:00 mysqlbackup: INFO: Copying /var/lib/mysql/mysql/slave_worker_info.ibd (Antelope file format).
141204 12:55:00 mysqlbackup: INFO: Copying /var/lib/mysql/test2/tb1.ibd (Antelope file format).
141204 12:55:00 mysqlbackup: INFO: Completing the copy of innodb files.
141204 12:55:00 mysqlbackup: INFO: Starting to copy Binlog files...
141204 12:55:01 mysqlbackup: INFO: Preparing to lock tables: Connected to mysqld server.
141204 12:55:01 mysqlbackup: INFO: Starting to lock all the tables...
141204 12:55:02 mysqlbackup: INFO: All tables are locked and flushed to disk
141204 12:55:02 mysqlbackup: INFO: Completed the copy of binlog files...
141204 12:55:02 mysqlbackup: INFO: Opening backup source directory '/var/lib/mysql/'
141204 12:55:02 mysqlbackup: INFO: Starting to backup all non-innodb files in
 subdirectories of '/var/lib/mysql/'
141204 12:55:02 mysqlbackup: INFO: Adding database directory: datadir/mysql
141204 12:55:02 mysqlbackup: INFO: Adding database directory: datadir/performance_schema
141204 12:55:02 mysqlbackup: INFO: Completing the copy of all non-innodb files.
141204 12:55:02 mysqlbackup: INFO: Adding database directory: datadir/test
141204 12:55:02 mysqlbackup: INFO: Adding database directory: datadir/test2
141204 12:55:04 mysqlbackup: INFO: A copied database page was modified at 1631766.
          (This is the highest lsn found on page)
          Scanned log up to lsn 1631766.
          Was able to parse the log up to lsn 1631766.
          Maximum page number for a log record 0
141204 12:55:04 mysqlbackup: INFO: All tables unlocked
141204 12:55:04 mysqlbackup: INFO: All MySQL tables were locked for 2.057 seconds.
141204 12:55:04 mysqlbackup: INFO: Reading all global variables from the server.
141204 12:55:04 mysqlbackup: INFO: Completed reading of all global variables from the server.
141204 12:55:04 mysqlbackup: INFO: Creating server config files server-my.cnf and server-all.cnf in /home/admi
 mysqlbackup: INFO: Copying meta file /home/admin/backup-tmp/meta/backup_variables.txt.
 mysqlbackup: INFO: Copying meta file /home/admin/backup-tmp/datadir/ibbackup_logfile.
 mysqlbackup: INFO: Copying meta file /home/admin/backup-tmp/server-all.cnf.
 mysqlbackup: INFO: Copying meta file /home/admin/backup-tmp/server-my.cnf.
 mysqlbackup: INFO: Copying meta file /home/admin/backup-tmp/meta/backup_content.xml.
 mysqlbackup: INFO: Copying meta file /home/admin/backup-tmp/meta/image_files.xml.
141204 12:55:08 mysqlbackup: INFO: Full Image Backup operation completed successfully.
141204 12:55:08 mysqlbackup: INFO: Backup image created successfully.
 mysqlbackup: INFO: Image Path = /home/admin/backups/my.mbi
141204 12:55:08 mysqlbackup: INFO: MySQL binlog position: filename mysqld-bin.000004, position 120

------------------------------------------------------------
   Parameters Summary
------------------------------------------------------------
   Start LSN                  : 1631744
   End LSN                    : 1631766
------------------------------------------------------------

mysqlbackup completed OK!
```

4.2.3 Verifying a Backup

To verify that your backup has been successful, restore the backup data on a different server and run the MySQL daemon (`mysqld`) on the new data directory. You can then execute `SHOW` statements to verify the database and table structures, and execute queries to verify further details of the database.

See Section 4.2.4, “Restoring a Database” for basic steps for restoring a backup, and see Chapter 5, *Recovering or Restoring a Database* for more detailed instructions. 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 of the backed up MySQL instance, combined with the settings from the `backup-my.cnf` file, which was created in the temporary directory you specified with `--backup-dir` when you created a single-image backup (see Section 4.2.2, “Backing Up an Entire MySQL Instance”) and contains a small subset of parameters required by `mysqlbackup`. Create a new configuration file by concatenating the two files mentioned above into a new one, and use that file on the server on which you perform the verification. Edit the file to make sure the `datadir` parameter points to the right location on the verification server. Edit the values for port, socket, and so on if you need to use different connection settings on the verification server.

4.2.4 Restoring a Database

To restore a MySQL instance from a backup to a database server:

- Shut down the database server.
- Delete all files inside the server's data directory. Also delete all files inside the directories specified by the `--innodb_data_home_dir`, `--innodb_log_group_home_dir`, and `--innodb_undo_directory` options for restore, if the directories are different from the data directory.
- Use, for example, the `copy-back-and-apply-log` command, which converts the raw backup into a prepared backup by updating it to a consistent state, and then copies the tables, indexes, metadata, and any other required files onto a target server. For the various options that you can specify for this operation, see Section 13.3, “Restore Operations”.

In the example below, the single-file backup created in the example given in Section 4.2.2, “Backing Up an Entire MySQL Instance” is restored using the `copy-back-and-apply-log` command. Besides the usual connection parameters, the following options are used:

- `--defaults-file` supplies the configuration for restoring the data. It must be the first option to appear in a `mysqlbackup` command, if ever used. In most cases, you can supply to `mysqlbackup` with this option the configuration file for the target server to which you are restoring the data. However, when the following InnoDB settings for the backup are different from those on the target server, it is important to supply the values for the backup to `mysqlbackup` during restore and to `mysqld` when you start the restored server (otherwise, the restore might fail, or you might have problem starting the restored server afterwards):
  - `innodb_data_file_path`
  - `innodb_log_file_size`
  - `innodb_log_files_in_group`
  - `innodb_page_size`
  - `innodb_checksum_algorithm`

  If you are not sure about those settings for your backup, they are stored in the `backup-my.cnf` file during the backup—you can find the file either in the temporary directory you specified with `--backup-dir` when you created the single-image backup, or in a backup directory you can create by unpacking the backup image using the `extract` command. If the values of these options differ from those on the target server, add them to the configuration file you are supplying to `mysqlbackup` and also to the configuration file you are going to use to start the server afterwards; alternatively, you can also supply them as command line options to `mysqlbackup` and `mysqld`.

For some of the options listed above (namely, `innodb_data_file_path`, `innodb_log_file_size`, and `innodb_log_files_in_group`), `mysqlbackup` checks the values you supply for them to ensure that you will be able to start the target server afterwards with those values: it throws an error if any of them does not match with the actual values for the backup. Warnings are given if those values are not specified for `mysqlbackup` in either the configuration file or on the command line (which is the case in the example below).

- `--datadir` supplies the location of the data directory for restoring the data. You must specify this option for any restore operation.
- `--backup-image` provides the path of the single-file backup.
- `--backup-dir` provides the location of an empty folder to store some temporary files created during the restore procedure.

```
$  ./mysqlbackup  --defaults-file=/etc/mysql/my.cnf --datadir=/var/lib/mysql \
    --backup-image=/home/admin/backups/my.mbi --backup-dir=/home/admin/backup-tmp copy-back-and-apply-log

MySQL Enterprise Backup version 3.12.5 Linux-2.6.18-274.el5-i686 [2014/11/12]
Copyright (c) 2003, 2014, Oracle and/or its affiliates. All Rights Reserved.

 mysqlbackup: INFO: Starting with following command line ...
 ./mysqlbackup --defaults-file=/etc/mysql/my.cnf
        --datadir=/var/lib/mysql --backup-image=/home/admin/backups/my.mbi
        --backup-dir=/home/admin/backup-tmp copy-back-and-apply-log

 mysqlbackup: INFO:
IMPORTANT: Please check that mysqlbackup run completes successfully.
           At the end of a successful 'copy-back-and-apply-log' run mysqlbackup
           prints "mysqlbackup completed OK!".

 mysqlbackup: INFO: Backup Image MEB version string: 3.12.5 [2014/11/12]
141204 13:10:39 mysqlbackup: INFO: MEB logfile created at /home/admin/backup-tmp/meta/MEB_2014-12-04.13-10-39_

--------------------------------------------------------------------
                      Server Repository Options:
--------------------------------------------------------------------
  datadir = /var/lib/mysql
  innodb_data_home_dir = /var/lib/mysql
  innodb_data_file_path = ibdata1:12M:autoextend
  innodb_log_group_home_dir = /var/lib/mysql
  innodb_log_files_in_group = 2
  innodb_log_file_size = 50331648
  innodb_page_size = 16384
  innodb_checksum_algorithm = innodb

--------------------------------------------------------------------
                      Backup Config Options:
--------------------------------------------------------------------
  datadir = /home/admin/backup-tmp/datadir
  innodb_data_home_dir = /home/admin/backup-tmp/datadir
  innodb_data_file_path = ibdata1:12M:autoextend
  innodb_log_group_home_dir = /home/admin/backup-tmp/datadir
  innodb_log_files_in_group = 2
  innodb_log_file_size = 50331648
  innodb_page_size = 16384
  innodb_checksum_algorithm = innodb

 mysqlbackup: INFO: Creating 14 buffers each of size 16777216.
141204 13:10:39 mysqlbackup: INFO: Copy-back-and-apply-log operation starts with following threads
  1 read-threads    6 process-threads    1 write-threads
141204 13:10:39 mysqlbackup: INFO: Copying database directory: meta
141204 13:10:39 mysqlbackup: INFO: Copying datadir/ibdata1.
```

```
141204 13:10:39 mysqlbackup: INFO: Copying datadir/mysql/innodb_index_stats.ibd.
141204 13:10:39 mysqlbackup: INFO: Copying datadir/mysql/innodb_table_stats.ibd.
141204 13:10:39 mysqlbackup: INFO: Copying datadir/mysql/slave_master_info.ibd.
141204 13:10:39 mysqlbackup: INFO: Copying datadir/mysql/slave_relay_log_info.ibd.
141204 13:10:39 mysqlbackup: INFO: Copying database directory: datadir/mysql
141204 13:10:39 mysqlbackup: INFO: Copying datadir/mysql/slave_worker_info.ibd.
141204 13:10:39 mysqlbackup: INFO: Copying datadir/test2/tb1.ibd.
141204 13:10:39 mysqlbackup: INFO: Copying database directory: datadir/performance_schema
141204 13:10:39 mysqlbackup: INFO: Copying database directory: datadir/test
141204 13:10:39 mysqlbackup: INFO: Copying database directory: datadir/test2
141204 13:10:40 mysqlbackup: INFO: Copying database directory: datadir/mysql
141204 13:10:41 mysqlbackup: INFO: Copying database directory: datadir/performance_schema
141204 13:10:42 mysqlbackup: INFO: Copying database directory: datadir/test
141204 13:10:42 mysqlbackup: INFO: Copying database directory: datadir/test2
141204 13:10:43 mysqlbackup: INFO: Total files as specified in image: 161
141204 13:10:43 mysqlbackup: INFO: Creating server config files server-my.cnf and server-all.cnf in /var/li
141204 13:10:43 mysqlbackup: INFO: Copy-back operation completed successfully.
 mysqlbackup: INFO: Source Image Path = /home/admin/backups/my.mbi


 mysqlbackup: INFO: Creating 14 buffers each of size 65536.
141204 13:10:43 mysqlbackup: INFO: Apply-log operation starts with following threads
  1 read-threads    1 process-threads
 mysqlbackup: INFO: Using up to 100 MB of memory.
141204 13:10:43 mysqlbackup: INFO: ibbackup_logfile's creation parameters:
          start lsn 1631744, end lsn 1631766,
          start checkpoint 1631766.
InnoDB: Doing recovery: scanned up to log sequence number 1631766
 mysqlbackup: INFO: InnoDB: Starting an apply batch of log records to the database...
InnoDB: Progress in percent: 0 1 2 3 4 5 6 7 8 9 10 11 12 13 14 15 16 17 18 19 20 21 22 23 24 25 26 27 28 2
 mysqlbackup: INFO: InnoDB: Setting log file size to 50331648
 mysqlbackup: INFO: InnoDB: Setting log file size to 50331648
141204 13:10:44 mysqlbackup: INFO: We were able to parse ibbackup_logfile up to
          lsn 1631766.
 mysqlbackup: INFO: Last MySQL binlog file position 0 120, file name mysqld-bin.000004:120
141204 13:10:44 mysqlbackup: INFO: The first data file is '/var/lib/mysql/ibdata1'
          and the new created log files are at '/var/lib/mysql'
141204 13:10:44 mysqlbackup: INFO: Apply-log operation completed successfully.
141204 13:10:44 mysqlbackup: INFO: Full Backup has been restored successfully.

mysqlbackup completed OK!
```

Now the original database directory is restored from the backup. Depending on how you are going to start the restored server, you might need to adjust the ownership of the restored data directory. For example, if the server is going to be started by the user `mysql`, use the following command to change the owner attribute of the data directory and the files under it to the `mysql` user, and the group attribute to the `mysql` group.

```
$ chown -R mysql:mysql /path/to/datadir
```

You are now ready to start the restored database server. For more discussions on how to perform different kinds of restores, see Section 5.2, “Performing a Restore Operation”.

4.3 Backup Scenarios and Examples

4.3.1 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 have created a full backup, you might perform incremental backups (which are smaller and faster) for the next several backup tasks. You then make a full backup periodically to begin the cycle again.

For sample commands for making a full backup, see Section 4.2.2, “Backing Up an Entire MySQL Instance”.

This section outlines some of the things to consider when deciding on a strategy for creating full backups. As we shall see, factors like speed, capacity, and convenience are all relevant for your decisions.

Options on Command Line or in Configuration File?

For clarity, the examples in this manual often show some of the command-line options that are used with the `mysqlbackup` commands. For convenience and consistency, you can include those options that remain unchanged for most backup jobs into the `[mysqlbackup]` section of the MySQL configuration file that you supply to `mysqlbackup`. `mysqlbackup` also picks up the options from the `[mysqld]` section if they are present there. Putting the options into a configuration file can simplify backup administration for you: for example, putting port information into a configuration file, you can avoid the need to edit your backup scripts each time the database instance switches to a different port. See Chapter 15, *Configuration Files and Parameters* for details about the use of configuration files.

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. The timestamped subdirectories make it simpler to establish retention periods, allowing easy removal and archiving of backup data that has passed 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` option to overwrite existing backup files.

For incremental backups that uses the `--incremental-base` option to specify the directory containing the previous backup, in order to make the directory names predictable, you might prefer to not use the `--with-timestamp` option and generate a sequence of directory names with your backup script instead .

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 want to run a full backup each time. However, you can usually save time and storage space by running periodic full backups and then several incremental backups in between them, as described in Section 4.3.2, “Making a Differential or Incremental Backup”.

Use Compression or Not?

Creating a compressed backup can save you considerable storage space and reduce I/O usage significantly. And with the LZ4 compression method (introduced since release 3.10), the overhead for processing compression is quite low. In cases where database backups are moving from a faster disk system where the active database files sit to a possibly slower storage, compression will often significantly lower the overall backup time. It can result in reduced restoration time as well. In general, we recommend LZ4 compression over no compression for most users, as LZ4-based backups often finish in a shorter time period. However, test out MySQL Enterprise Backup within your environment to determine what is the most efficient approach.

4.3.2 Making a Differential or Incremental Backup

Assuming a good portion of the data on your MySQL server remains unchanged over time, you can increase the speed and reduce the required storage space for your regular backups by backing up not all the data on the server each time, but only the changes to the data which have taken place over time. In order to that, after making first a full backup that contains all data, you can do one of the following:

- **Performing a series of differential backups.** Each differential backup includes all the changes made to the data since the last full backup was performed. To restore data up to, for example, time `t`, you simply restore first the full backup, and then, on top of it, the differential backup taken for time `t`.

- **Perform a series of incremental backup.** Each incremental backup only includes the changes since the previous backup, which can itself be a full or incremental backup. The first backup in an incremental series is always then a differential backup; but after that, each incremental backup only contains the changes made since that last incremental backup. Each subsequent incremental backup is thus usually smaller in size than a differential backup, and is faster to make; that allows you to make very frequent incremental backups, and then enables you to restore the database to a more precise point in time when necessary. However, restoring data with incremental backups might take longer and more work: in general, to restore data up to, for example, time `t`, you start with restoring the full backup, and then restore the incremental backups one by one, until you are finished with the incremental backup taken for time t.

MySQL Enterprise Backup supports both incremental and differential backups. You should decide on which backup strategy to adopt by looking at such factors like how much storage space you have, how quickly you have to be able to restore data, and so on.

MySQL Enterprise Backup treats differential backup as a special case of incremental backup that has a full backup as its base. To create a differential backup, simply follow the instructions below for performing incremental backups, and make sure you specify a full backup as the base of your incremental backup using the methods we describe below; you should also ignore any instructions that only apply to the handling of multiple incremental backups.

See Section 14.7, “Incremental Backup Options”, for descriptions of the `mysqlbackup` options used for incremental backups. An Incremental backup is enabled with one of the two options: `--incremental` and `--incremental-with-redo-log-only` option. See Creating Incremental Backups Using Only the Redo Log for their differences.

When creating an incremental backup, you have to indicate to `mysqlbackup` 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 on the server. Or, you can specify an explicit LSN value using the `--start-lsn` option, providing to `mysqlbackup` the ending LSN from a previous full or incremental backup (see Other Considerations for Incremental Backups on some limitation that applies when using the `--start-lsn` option).

To prepare the backup data to be restored, you combine all incremental backups 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.

Creating Incremental Backups Using Only the Redo Log

The `--incremental-with-redo-log-only` might offer some benefits over the `--incremental` option for creating an incremental backup:

- 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 dictated by 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 which case `mysqlbackup` will quickly determine that it cannot proceed and will return an error. Your backup script should be able to catch that error and then perform an incremental backup with the `--incremental` option instead.

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 the value of `innodb_log_files_in_group`. To compute the redo log size at the physical level, look into the `datadir` directory of the MySQL instance and sum up 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 a long startup time when the MySQL server was killed rather than shut down normally. With MySQL 5.5 and higher, the performance of crash recovery is significantly improved, as described in Optimizing InnoDB Configuration Variables, so that 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, it is recommended that you always use 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 decide 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, in order to avoid the likelihood of the backup failing because the historical data are 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 when 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, the amount of DML activity, and the size of your redo log files. Do your testing on a server with a realistic data volume and 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.

Other 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. 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.

For non-InnoDB files, the entire file is included in an incremental backup if that file has changed since the previous backup, which means the savings for backup resources are less significant when comparing with the case with InnoDB tables.

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

When making an incremental backup that is based on a backup (full or incremental) created using the `--no-locking` option, use the `--skip-binlog` option to skip the backing up of the binary log, as binary log information will be unavailable to `mysqlbackup` in that situation.

No binary log files are copied into the incremental backup if the `--start-lsn` option is used. To include binary log files for the period covered by the incremental backup, use the `--incremental-base` option instead, which provides the necessary information for `mysqlbackup` to ensure that no gap exists between binary log data included in the previous backup and the current incremental backup.

Examples of Incremental Backups

This example uses `mysqlbackup` 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 can just specify the location of the previous directory 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 want to 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/dbadmin/.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!
```

Note that if your last backup was a single-file instead of a directory backup, you can still use `--incremental-base` by specifying for `dir:`*`directory_path`* the location of the temporary directory you supplied with the `--backup-dir` option during the full backup.

As an alternative to specifying `--incremental-base=dir:`*`directory_path`*, you can tell `mysqlbackup` to query the `end_lsn` value from the last successful backup as recorded in the `backup_history` table on the server using `--incremental-base=history:last_backup` (this required that the last backup was made with `mysqlbackup` connected to the server).

You can also use the `--start-lsn` option to specify where the incremental backup should start. You have to record the LSN of the previous backup reported by `mysqlbackup` at the end of the backup:

```
mysqlbackup: Was able to parse the log up to lsn 2654255716
```

The number is also recorded in the `meta/backup_variables.txt` file in the folder specified by `--backup-dir` during the backup. Supply then that number to `mysqlbackup` using the `--start-lsn` option. The incremental backup then includes all changes that came *after* the specified LSN. Since then the location of the previous backup is not very significant then, you can use `--with-timestamp` to create named subdirectories automatically.

```
$ mysqlbackup --defaults-file=/home/dbadmin/.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!
```

To create an incremental backup image instead, use the following command, specifying with `--backup-dir` a temporary directory for storing the metadata for the backup and some temporary files:

```
$ mysqlbackup --defaults-file=/home/dbadmin/.my.cnf --incremental \
  --start-lsn=2654255716 \
  --with-timestamp \
  --backup-dir=/incr-tmp \
  --backup-image=/incr-backup/incremental_image.bi \
  backup-to-image
```

In the following example though, because `--backup-image` does not provide a full path to the image file to be created, the incremental backup image is created under the folder specified by `--backup-dir`:

```
$ mysqlbackup --defaults-file=/home/dbadmin/.my.cnf --incremental \
  --start-lsn=2654255716 \
  --with-timestamp \
  --backup-dir=/incr-images \
  --backup-image=incremental_image1.bi \
  backup-to-image
```

On how to restore your database using the incremental backups, see Section 5.2.3, “Restoring an Incremental Backup”

Next steps:

- On a regular schedule determined by date or amount of database activity, take more 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.

4.3.3 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 or save transmission

time when sending the backup data to another server. Also, compression often results in faster backups because of reduced IO.

The backup compression feature works only for InnoDB tables. After the InnoDB tablespace files are compressed during backup, they receive the `.ibz` extension. To avoid wasting CPU cycles without saving additional disk space, `--compress` does not attempt to compress tables that were already-compressed on the server (see Enabling Compression for a Table); nevertheless, such tablespace files are also saved with the `.ibz` extension inside the backup.

Note

When there is unused space within an InnoDB tablespace file, the entire file is copied during an uncompressed backup. Perform a compressed backup to avoid the storage overhead for the unused space.

The binary log and relay log files are compressed and saved with the `.bz` extension when being included in a compressed backup.

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

You can also select the compression algorithm to use by the `--compress-method` option and, when using the ZLIB or LZMA compression algorithm, the level of compression by the `--compress-level` option. See Section 14.6, “Compression Options” for details.

This is a sample command for making a compressed directory backup:

```
mysqlbackup --defaults-file=/etc/my.cnf --compress backup
```

This is a sample command for making a compressed and prepared directory backup (only supported for MySQL Enterprise Backup 3.12.3 and later):

```
mysqlbackup --defaults-file=/etc/my.cnf --compress-method=zlib --compress-level=5 backup-and-apply-log
```

This is a sample command for making a compressed single-file backup:

```
mysqlbackup --defaults-file=/etc/my.cnf --compress-method=zlib --compress-level=5 \
  --backup-image=backup.img backup-to-image
```

Note

See the limitation that applies to compressed backups in Appendix B, *Limitations of MySQL Enterprise Backup*.

Next steps:

- Make a note of the LSN value in the message at the end of both the full and incremental backups (for example, in the line `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 smaller and can be made faster than a full backup.

4.3.4 Making a Partial Backup

Note

To facilitate the creation of partial backups, MySQL Enterprise Backup has introduced two new options for partial backup since version 3.10: `--include-tables` and `--exclude-tables`. The new options are intended for replacing the older options of `--include`, `--databases`, `--databases-list-file`, and `--only-innodb-with-frm`, which are incompatible with the new options and will be deprecated in the upcoming releases. In the discussions below we assume the new options are used for partial backups. For reference purpose, we have included information on the older options at the end of this section in Making a Partial Backup with Legacy Options.

By default, all the files under the database subdirectories in the data directory are included in the backup, so that 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.

There are various ways to create different kinds of partial backup with MySQL Enterprise Backup:

- Including or excluding specific tables by their names. This uses the `--include-tables` or `--exclude-tables` option.

  Each table is checked against the regular expression specified with the `--include-tables` or `--exclude-tables` option. If the regular expression matches the fully qualified name of the table (in the form of `db_name.table_name`), the table is included or excluded for the backup. The regular expression syntax used is the extended form specified in the POSIX 1003.2 standard. The options have been implemented with the RE2 regular expression library.

- Including some or all InnoDB tables, but not other table types. This uses the `--only-innodb` option.

- Leaving out files that are present in the MySQL data directory but not actually part of the MySQL instance. This uses the `--only-known-file-types` option.

- Achieving a multiple of selection effects by using a combination of the above mentioned options.

- Backing up a selection of InnoDB tables using transportable tablespaces (TTS). This uses the `--use-tts` and the `--include-tables` or `--exclude-tables` (or both) options.

For syntax details on all the options involved, see Section 14.8, “Partial Backup and Restore Options”.

Important

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.

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 the other databases. Always restore partial backups on a fresh MySQL server instance without any other InnoDB tables that you want to preserve.

The following are some command samples for partial backups.

Including all tables with names starting with “emp” into the backup:

```
$ mysqlbackup \
 --host=localhost --user=mysqluser --protocol=TCP --port=3306 \
 --backup-dir=$MEB_BACKUPS_DIR/backupdir \ --include-tables="\.emp" \
 backup
```

Taking a backup of all tables except tables from the “mysql” and “performance_schema” databases:

```
$ mysqlbackup \
 --host=localhost --user=mysqluser --protocol=TCP --port=3306 \
 --backup-dir=$MEB_BACKUPS_DIR/backupdir \
 --exclude-tables="^(mysql|performance_schema)\." \
 backup
```

Taking a backup of all tables in the “sales” database, but excludes the table with the name “hardware”

```
$ mysqlbackup \
 --host=localhost --user=mysqluser --protocol=TCP --port=3306 \
 --backup-dir=$MEB_BACKUPS_DIR/backupdir \
 --include-tables="^sales\." --exclude-tables="^sales\.hardware$" \
 backup
```

Taking a backup of all tables in the “sales reps” database, but excludes the table with the name “euro-asia” (special characters like spaces or dashes are supported by the partial backup options since release 3.12.1):

```
$ mysqlbackup \
 --host=localhost --user=mysqluser --protocol=TCP --port=3306 \
 --backup-dir=$MEB_BACKUPS_DIR/backupdir \
 --include-tables="^sales reps\." --exclude-tables="^sales reps\.euro-asia" \
 backup
```

Backing up all InnoDB tables, but not `.frm` files:

```
$ mysqlbackup --defaults-file=/home/dbadmin/.my.cnf --only-innodb backup
```

You can also make compressed, single-image, and other kinds of selective backups by using the appropriate command options.

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 these ones.

Making a Partial Backup with the Legacy Options

Important

Information in this subsection is only for using the legacy options of `--include`, `--databases`, `--databases-list-file`, and `--only-innodb-with-frm`, which will be deprecated in the upcoming issues. For creating partial backups, it

is strongly recommended that the new options of `--include-tables` and `--exclude-tables` be used instead. Note that you cannot combine the legacy and the new partial-backup options in a single command.

MySQL Enterprise Backup can make different kinds of partial backup using the legacy partial-backup options:

- 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 Legacy 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.

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 syntax used is the extended form specified in the POSIX 1003.2 standard. On Unix-like systems, quote the regular expression appropriately to prevent interpretation of shell meta-characters. This feature has been implemented with the RE2 regular expression library.

The backup directory produced contains a backup log file and copies of InnoDB data files.

IMPORTANT: Although `mysqlbackup` 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 also use them together with the `--include` option to make selective backup of InnoDB tables while excluding all other files created by other storage engines.

Example 4.1 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 data files (`.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/dbadmin/.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/dbadmin/.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 4.2 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/dbadmin/.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 data files.

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

The `--databases` and `--databases-list-file` options of `mysqlbackup` 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 you 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.

4.3.5 Making a Single-File Backup

To avoid having a large number of backup files to keep track, store, and transport, `mysqlbackup` can create a backup in a single-file format. It can also 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 into a directory tree. For the syntax of the relevant `mysqlbackup` options, see Section 14.9, “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, you can use the technique to put the backup onto another storage device or server and avoid significant storage overhead on the original database server.

To create a single-file backup, use the `backup-to-image` command. The following examples illustrate how to perform a single-file backup and other related operations.

Example 4.3 Single-File Backup to Absolute Path

This command creates a single backup image on 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 4.4 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 4.5 Single-File Backup to Standard Output

The following command dumps the backup output to standard output. Again, the folder specified with the `--backup-dir` option is used as a temporary directory.

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

Example 4.6 Convert Existing Backup Directory to Single Image

The `backup-dir` directory is bundled into the `/backup/my.mbi` file.

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

Example 4.7 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 4.8 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 4.9 Validate a Single-File Backup

The following command verifies that the single-file backup is not corrupted, truncated, or damaged by validating the checksum value for each data page in the backup.

```
mysqlbackup --backup-image=/logs/fullimage.mi  validate
```

Example 4.10 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 4.11 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 4.12 Selective Extract of Single File

The following command extracts the single file `meta/comments.txt` from the backup image `my.mbi` 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 from the backup image `my.mbi` 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` (which is inside the single-file backup `my.mbi`) to standard output.

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

Example 4.13 Selective Extract of Single Directory

The following command extracts a single directory `meta` from the backup image `my.mbi` into a local file system path `./meta`. All contents in the `meta` directory are extracted, including any subdirectories.

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

The following command extracts all contents of the `meta` directory, including 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 4.14 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.

4.3.5.1 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. You can achieve that with a single-file backup. To send the single-file backup to standard output, use the `mysqlbackup` command `backup-to-image` without specifying the `--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`, and so on, which take the 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 `mysqlbackup` with the `image-to-backup-dir` option on the other end to reproduce the directory structure of a regular backup.

Example 4.15 Single-File Backup to a Remote Host

The following command streams the backup as a single-file output to a remote host to be saved under the file name `my_backup.img` (`--backup-dir=/tmp` designates the directory for storing temporary files rather than the final output file):

```
mysqlbackup --defaults-file=~/my_backup.cnf --backup-image=- --backup-dir=/tmp backup-to-image | \
  ssh <user name>@<remote host name> 'cat > ~/backups/my_backup.img'
```

For simplicity, all the connection and other necessary options are assumed to be specified in the default configuration file. `ssh` can be substituted with another communication protocol like `ftp`, and `cat` can be substituted with another command (for example, `dd` or `tar` for normal archiving).

Example 4.16 Single-file Backup to a Remote MySQL Server

The following command streams the backup as a single backup file to be restored on a remote MySQL server:

```
mysqlbackup  --backup-dir=backup --backup-image=-  --compress backup-to-image | \
ssh <user name>@<remote host name> 'mysqlbackup --backup-dir=backup_tmp --datadir=/data \
--innodb_log_group_home_dir=. \
--innodb_log_files_in_group=<innodb_log_files_in_group_of_backedup_server> \
 --innodb_log_file_size=<innodb_log_file_size_of_backedup_server> \
--innodb_data_file_path=<innodb_data_file_path_of_backedup_server> \
--uncompress --backup-image=- copy-back-and-apply-log'
```

Example 4.17 Stream a Backup Directory to a Remote MySQL Server

The following command streams a backup directory as a single backup file to be restored on a remote MySQL server:

```
mysqlbackup --backup-image=- --backup-dir=/path/to/my/backup backup-dir-to-image | \
ssh <user name>@<remote host name> 'mysqlbackup --backup-dir=backup_tmp --datadir=/data --backup-image=- c
```

4.3.5.2 Backing Up to Tape

Tape drives are affordable, high-capacity storage devices for backup data. MySQL Enterprise Backup 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 9, *Using MySQL Enterprise Backup with Media Management Software (MMS) Products*.

4.3.5.3 Backing Up to Cloud Storage

MySQL Enterprise Backup supports cloud backups. Only single-file backups can be created on and restored from a cloud storage. All `mysqlbackup` options compatible with single-file operations (including, for example, the incremental, compression, partial, and encryption options) can be used with cloud backups or restores.

Note

See Appendix B, *Limitations of MySQL Enterprise Backup* for some limitations regarding the support for cloud storage by `mysqlbackup`.

Currently, MySQL Enterprise Backup supports two types of cloud storage services: OpenStack Swift or compatible object storage services (for example, Oracle Cloud Infrastructure Object Storage and Oracle Cloud Infrastructure Object Storage Classic) and Amazon S3.

Note

Due to some issues, Amazon S3 is currently not supported by MySQL Enterprise Backup 3.12.

MySQL Enterprise Backup 3.12 supports the Swift v1.0 API, and also the OpenStack Identity (Keystone) API v2.0 for authentication. It also supports authentication using Swift's TempAuth system or HTTP Basic Authentication. Backups are stored as dynamic large objects in Swift, with each backup larger than 5G being split into multiple parts with names in the form of *<object_name>*_part_*<number>*. See the OpenStack documentation for details.

A cloud backup is created using the cloud options for `mysqlbackup`, which are described in details in Section 14.14, “Cloud Storage Options”. Here are some sample commands for creating a cloud backup:

Example 4.18 Creating a Cloud Backup on Oracle Cloud Infrastructure Object Storage Classic

This example creates a cloud backup in an Oracle Cloud Infrastructure (OCI) Object Storage Classic container, using the TempAuth system for authenticating the user's credentials.

```
mysqlbackup \
--cloud-service=openstack  --cloud-container=<OCI Object Storage Classic container> \
--cloud-user-id=<serviceInstanceName>-<identityDomainName>:<userName> --cloud-password='<password>' \
--cloud-tempauth-url=https://<dataCenterCode>.storage.oraclecloud.com \
--cloud-trace=1 --cloud-object=image_900.mbi \
--backup-dir=/home/user/dba/orbackuptmpdir \
--backup-image=- \
backup-to-image
```

Example 4.19 Creating a Cloud Backup on Oracle Cloud Infrastructure Object Storage

This example creates a cloud backup in an Oracle Cloud Infrastructure (OCI) Object Storage bucket, using HTTP basic authentication.

```
mysqlbackup \
--cloud-service=openstack  --cloud-container=<OCI Object Storage bucket> \
--cloud-user-id=<OCI userName> --cloud-password='<OCI auth token>' \
--cloud-ca-info=/etc/ssl/certs/ca-certificates.crt
--cloud-basicauth-url=https://swiftobjectstorage.<region>.oraclecloud.com/v1/<OCI Object Storage namespace> \
--cloud-trace=1 --cloud-object=backup_image_900.mbi \
--backup-dir=/home/user/dba/orbackuptmpdir \
--backup-image=- \
backup-to-image
```

Example 4.20 Creating a Cloud Incremental Backup on Oracle Cloud Infrastructure Object Storage

This example creates an incremental cloud backup in an Oracle Cloud Infrastructure (OCI) Object Storage bucket, using HTTP basic authentication.

```
mysqlbackup \
--cloud-service=openstack  --cloud-container=<OCI Object Storage bucket> \
--cloud-user-id=<OCI userName> --cloud-password='<OCI auth token>' \
--cloud-ca-info=/etc/ssl/certs/ca-certificates.crt
--cloud-basicauth-url=https://swiftobjectstorage.<region>.oraclecloud.com/v1/<OCI Object Storage namespace> \
--cloud-object=backup_incr_image_900.mbi \
--backup-dir=/home/user/dba/orincrbackuptmpdir \
--incremental --incremental-base=history:last_backup \
--backup-image=- \
backup-to-image
```

Example 4.21 Creating a Cloud Backup on an OpenStack Object Storage

This example creates a cloud backup on an OpenStack object storage, using the Keystone identity service to authenticate the user's credentials.

```
mysqlbackup \
--include-tables=testdb.t1 --use-tts=with-full-locking \
--cloud-service=openstack  --cloud-container=<swift container> \
--cloud-user-id=<keystone user> --cloud-password=<keystone password> \
--cloud-region=<keystone region> --cloud-tenant=<keystone tenant> \
--cloud-identity-url=<keystone url>  \
--cloud-trace=1 --cloud-object=image_800.mbi \
--backup-dir=/home/user/dba/opbackuptmpdir \
--backup-image=- \
backup-to-image
```

Example 4.22 Creating a Cloud Backup on Amazon S3

```
mysqlbackup\
--cloud-service=s3 --cloud-aws-region=<aws region> \
--cloud-access-key-id=<aws access key id> --cloud-secret-access-key=< aws secret access key> \
--cloud-bucket=<s3 bucket name> --cloud-object-key=<aws object key> \
--backup-dir=/home/user/dba/s3backuptmpdir \
--backup-image=- \
backup-to-image
```

A cloud backup always uses one write thread.

Besides `backup-to-image`, all other `mysqlbackup` operations for single-file backups (`backup-dir-to-image`, `list-image`, `validate`, `image-to-backup-dir`, `extract`, `copy-back`, and `copy-back-and-apply-log`) can also be performed with cloud storage. For example:

Example 4.23 Extract an Existing Image from an Oracle Cloud Infrastructure Object Storage Classic Container to a Backup Directory

Extract a backup image from an Oracle Cloud Infrastructure Object Storage Classic container, using the `--backup-dir` option to specify the directory into which the image will be extracted:

```
mysqlbackup \
--cloud-service=openstack  --cloud-container=<OCI Object Storage Classic container> \
--cloud-user-id=<serviceInstanceName>-<identityDomainName>:<userName> --cloud-password=<password> \
--cloud-tempauth-url=https://<dataCenterCode>.storage.oraclecloud.com \
--cloud-object=image_930.mbi \
--backup-dir=/home/user/dba/orbackupdir \
--backup-image=- \
image-to-backup-dir
```

Example 4.24 Extract an Existing Image from Amazon S3 Cloud Storage to a Backup Directory

Extract a backup image from Amazon S3, using the `--backup-dir` option to specify the directory into which the image will be extracted:

```
mysqlbackup\
--cloud-service=s3 --cloud-aws-region=<aws region> \
--cloud-access-key-id=<aws access key id> --cloud-secret-access-key=< aws secret access key> \
--cloud-bucket=<s3 bucket name> --cloud-object-key=<aws object key> \
--backup-dir=/home/user/dba/s3backupdir \
--backup-image=- \
image-to-backup-dir
```

See Section 5.2.6, “Restoring a Backup from Cloud Storage to a MySQL Server” on how to restore a backup image from a cloud storage.

4.3.6 Making an Optimistic Backup

Optimistic backup is a feature introduced in MySQL Enterprise Backup 3.11 for improving performance for backing up and restoring huge databases in which only a small number of tables are modified frequently.

During a hot backup of a huge database (say, in the order of terabytes), huge redo log files could be generated on the server when the backup is in progress. As the redo log files grow faster than they can be processed by `mysqlbackup`, the backup operation can actually fail when `mysqlbackup` cannot catch up with the redo log cycles and LSNs get overwritten by the server before they are read by `mysqlbackup`. Moreover, the `apply-log` step for preparing a backup for restoration can take a very long time as `mysqlbackup` has huge `ibbackup_logfile` files (created from the big redo log files) to apply to the backup. The problems are intensified when the I/O resources available for reading and writing the redo logs are scarce during the backup and restoration processes.

Optimistic backup relieves the problems by dividing the backup process into two internal phases, which are transparent to the users:

1. Optimistic phase: In this first phase, tables that are unlikely to be modified during the backup process (referred to as the “inactive tables” below, identified by the user with the `optimistic-time` option or, by exclusion, with the `optimistic-busy-tables` option) are backed up without locking the MySQL instance. And because those tables are not expected to be changed before the backup is finished, redo logs, undo logs, and system table spaces are not backed up by `mysqlbackup` in this phase.

2. Normal phase: In this second phase, tables that are not backed up in the first phase (referred to as the “busy tables” below) are being backed up in a manner similar to how they are processed in an ordinary backup: the InnoDB files are copied first, and then other relevant files and copied or processed with the MySQL instance locked. The redo logs, undo logs, and the system tablespace are also backed up in this phase.

An optimistic backup occurs whenever the `optimistic-time` or `optimistic-busy-tables` option is used. For how to use the options, see detailed descriptions for them in Section 14.10, “Performance / Scalability / Capacity Options”. If, as expected, the list of inactive tables identified by the optimistic options do not change during the backup (or, even if it changes by a small percentage), most users will find that the overall backup time is reduced significantly compared to an ordinary backup, as the size of the redo log data to be backed up will be far smaller. Additionally, restore time for the backup will also be reduced, as the `apply-log` operation will be much faster because of the smaller redo log. However, if it turns out that the list of inactive tables identified changed by a significant portion during the backup process, benefits of performing an optimistic back up will become limited and, in the worst case, an optimistic backup might actually take longer to perform and, for a single-file backup, the size of the backup will be larger when comparing with an ordinary backup. Therefore, users should be careful in identifying which tables are “inactive” and which are “busy” when trying to perform an optimistic backup.

Note

An optimistic backup cannot be performed for an incremental backup or a backup using transportable tablespaces (TTS).

The following examples illustrate how to make an optimistic backup.

Example 4.25 Optimistic Backup Using the Option `optimistic-time=YYMMDDHHMMSS`

In this example, tables that have been modified since the noon of May 16, 2011 are treated as busy tables and backed up in the normal phase of an optimistic backup, and all other tables are backed up in the optimistic phase:

```
mysqlbackup --defaults-file=/etc/my.cnf --optimistic-time=110516120000 backup
```

Example 4.26 Optimistic Backup Using the Option `optimistic-time=now`

In this example, all tables are treated as inactive tables and backed up in the optimistic phase of an optimistic backup:

```
mysqlbackup --defaults-file=/etc/my.cnf --optimistic-time=now backup
```

Example 4.27 Optimistic Backup Using the `optimistic-busy-tables` Option

In this example, tables in `mydatabase` that are prefixed by `mytables-` in their names are treated as busy tables and backed up in the normal phase of an optimistic backup, and all other tables are backed up in the optimistic phase:

```
mysqlbackup --defaults-file=/etc/my.cnf --optimistic-busy-tables="^mydatabase\.mytables-.*"  backup
```

When you use both the `optimistic-time` and `optimistic-busy-tables` options and they come into conflict on determining which tables are to be busy tables, `optimistic-busy-tables` takes precedence over `optimistic-time`. For example:

Example 4.28 Optimistic and Partial Backup Using both the `optimistic-busy-tables` and `optimistic-time` Options

In this example, tables in `mydatabase` that are prefixed by `mytables-` in their names are treated as busy tables and backed up in the normal phase, even if they have not been modified since May 16, 2010, the time specified by `optimistic-time`:

```
mysqlbackup --defaults-file=/etc/my.cnf --optimistic-busy-tables="^mydatabase\.mytables-.*"  \
--optimistic-time=100516 backup
```

4.3.7 Making a Back Up of In-Memory Database Data

The `--exec-when-locked` option of `mysqlbackup` lets you specify a command (together with the desired command 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 option value within single quotes.

4.3.8 Making Scheduled Backups

Maintaining a regular backup schedule is an important measure for preventing data loss for you MySQL server. This section discusses some simple means for setting up a schedule for running MySQL Enterprise Backup.

For Linux and other Unix-like platforms: you can set up a cron job on your system for scheduled backups. There are two types of cron jobs. To set up a user cron job, which is owned and run by a particular user, do the following:

- Log on as the user who runs MySQL Enterprise Backup and use the following command to invoke an editor for creating (or modifying) a crontab:

  ```
  $> crontab -e
  ```

- In the editor, add an entry similar to the following one to the crontab, and then save your changes:

  ```
  @daily /path-to-mysqlbackup/mysqlbackup -uroot --backup-dir=/path-to-backup-folder/cronbackups --with-ti
  ```

  This crontab entry invokes `mysqlbackup` to create a backup under the `cronbackups` directory at `00:00:00` everyday. Outputs from the `stderr` and `stdout` streams are redirected to /dev/null/, so they will not invoke other actions on the part of the Cron server (for example, email notifications to the user).

To set up a system cron job, which is owned and run by `root`, create a file under the `/etc/cron.d` folder and put into it a similar crontab entry as the one above, adding the user (`root` in the following example) before the `mysqlbackup` command:

```
@daily root /path-to-mysqlbackup/mysqlbackup -uroot --backup-dir=/path-to-backup-folder/cronbackups --with
```

Check your platform's documentation for further details on the different ways to set up cron jobs for various types of schedules.

For Windows platforms: Use the Task Scheduler for the purpose. Check the documentation for your Windows platform for instructions.

4.4 Making Backups with a Distributed File System (DFS) or Storage Access Network (SAN)

When system administrators attempt to set up MySQL and MySQL Enterprise Backup in an environment that uses a distributed file system (DFS) or a storage access network (SAN), the MySQL server, the server's data directory, MySQL Enterprise Backup, and the backup directory may end up existing on different physical servers. When that happens, the operations of `mysqlbackup` might be impacted. The operation most likely to be adversely affected is hot backup, the success of which depends on:

1. Each page of a data file is copied consistently, that is, all the bytes in the page correspond to the same LSN.
2. No copied page is older than the time that marks the beginning of the temporal duration the backup is supposed to cover.
3. The redo log is copied consistently, meaning a continuous segment of redo log is copied, and it includes all the changes from the beginning of the temporal period that the backup is to cover until the end of the backup operation. Each block of the copied redo log has to be consistent.

Condition 1 is easily achievable with most DFSs or SANs of reasonable performance. Condition 2 though can remain unfulfilled even when condition 1 has been satisfied: for example, `mysqlbackup` could copy all the pages of a tablespace correctly except for one page for which `mysqlbackup` has included an old version into the copy. If the LSN of that old version of the page is smaller than the LSN first seen by `mysqlbackup` at the beginning of the backup process, the resulting backup will be defective. This example shows that `mysqlbackup` may have problem performing a hot backup unless it can see the writes to the file system being executed in the correct order, that is, the order in which the server executed them.

Regarding condition 3, unlike data file pages, redo log blocks are written sequentially, which means condition 3 is easier to fulfill than conditions 1 and 2. However, if `mysqlbackup` reaches the highest LSN in the copied data file pages before encountering the end of the redo log, the backup fails. A failure occurs also if `mysqlbackup` reads a corrupted log block at any time during the copying of the redo log. Both these failures can occur if `mysqlbackup` does not see the same history of the file system states as the MySQL server does.

Therefore, to use `mysqlbackup` with a DFS or SAN, it is important to make sure that `mysqlbackup` sees all the writes to the file system in the same order as the MySQL server does. The condition is most likely to be satisfied when `mysqlbackup` and the MySQL server are running on the same server node, and it is unlikely to be always fulfilled when it is otherwise.

Chapter 5 Recovering or Restoring a Database

Table of Contents

5.1 Preparing the Backup to be Restored 57
5.2 Performing a Restore Operation 58
5.2.1 Restoring a Compressed Backup 59
5.2.2 Restoring an Encrypted Backup Image 60
5.2.3 Restoring an Incremental Backup 60
5.2.4 Restoring Backups Created with the `--use-tts` Option 61
5.2.5 Restoring External InnoDB Tablespaces to Different Locations 62
5.2.6 Restoring a Backup from Cloud Storage to a MySQL Server 62
5.3 Point-in-Time Recovery 63
5.4 Restoring a Backup with a Database Upgrade or Downgrade 65

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 replica). This section describes the procedures to handle those 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 soon 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 takes place during a 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` option, which allows the data and log files created by the failed `apply-log` operation to be overwritten.

For simple directory backups (which are not compressed and non-incremental), you can combine the initial backup and the `apply-log` steps using the `backup-and-apply-log` command.

You can also perform `apply-log` and `copy-back` (which restores the prepared backup) with a single `copy-back-and-apply-log` command.

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 4.3.3, “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 described in Section 4.3.2, “Making a Differential or 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 will 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` commands to perform a restore operation are `copy-back-and-apply-log` and `copy-back` (for directory backup only; see Section 5.1, “Preparing the Backup to be Restored”). Normally, the restoration process requires the database server to be already shut down (or, at least not operating on the directory you are restoring the data to), except for restorations of backups created with the `--use-tts` option; see explanations below. The process 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.

Example 5.4 Shutting Down and Restoring a Database

```
mysqladmin --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. The table allows you to perform future incremental backups using the `--incremental-base=history:last_backup` option.

See Section 4.2.4, “Restoring a Database” for an explanation of the important options used in a restore operation like `--defaults-file`, `--datadir`, `--backup-image`, and `--backup-dir`.

Important

When performing a restore, make sure the target directories for restore data are all clean, containing no old or unwanted data files (this might require manual removal of files at the locations specified by the `--datadir`, `--innodb_data_home_dir`, `--innodb_log_group_home_dir`, and `--innodb_undo_directory` options). The same cleanup is not required for restoring backups created with the`--use-tts` option (in which case other requirements described in Section 5.2.4, “Restoring Backups Created with the `--use-tts` Option” apply though).

Before restoring a hot directory backup using the `copy-back` command, the backup has to be prepared and made consistent using the `apply-log` command. See Section 5.1, “Preparing the Backup to be Restored” for details. You can combine the `apply-log` and the `copy-back` operations (as well as a number of other operations, depending on the kind of backup you are restoring) into a single step by using the `copy-back-and-apply-log` option instead:

Example 5.5 Restoring a Backup Directory using `copy-back-and-apply-log`

```
mysqlbackup --defaults-file=<my.cnf> \
  --backup-dir=/export/backups/full \
    copy-back-and-apply-log
```

The same command is typically used for single-file backups to perform the same functions:

Example 5.6 Restoring a Single-file Backup using `copy-back-and-apply-log`

```
mysqlbackup --defaults-file=<my.cnf> -uroot --backup-image=<image_name> \
  --backup-dir=<backupTmpDir> --datadir=<restoreDir> copy-back-and-apply-log
```

The `--backup-dir` option specifies a temporary directory into which temporary output, status files, and backup metadata are be saved.

Note

Due to a known issue (Bug# 20485910), restoring a partial image backup created with MySQL Enterprise Backup 3.11 or earlier requires using the `--force` option.

The following subsections describe a number of different scenarios for restoring a backup.

5.2.1 Restoring a Compressed Backup

Restore a compressed directory backup at *`<backupDir>`* to *`<restoreDir>`* on the server using `copy-back-and-apply-log` and the `--uncompress` option:

Example 5.7 Restoring a Compressed Backup

```
mysqlbackup --defaults-file=<my.cnf> -uroot --backup-dir=<backupDir> --datadir=<restoreDir> \
  --uncompress copy-back-and-apply-log
```

To restore a compressed and prepared directory backup created with the `backup-and-apply-log` command (which is only supported for MySQL Enterprise Backup 3.12.3 and later), use the `copy-back` command and the `--uncompress` option:

```
mysqlbackup --defaults-file=<my.cnf> -uroot --backup-dir=<backupDir> --datadir=<restoreDir> \
  --uncompress copy-back
```

To restore a compressed backup image named *<image_name>*, use the `copy-back-and-apply-log` command and the `--uncompress` option, with the `--backup-dir` option specifying a temporary directory into which temporary output, status files, and backup metadata will be saved:

```
mysqlbackup --defaults-file=<my.cnf> -uroot --backup-image=<image_name> \
  --backup-dir=<backupTmpDir> --datadir=<restoreDir> --uncompress copy-back-and-apply-log
```

See Section 4.3.3, “Making a Compressed Backup” and Section 14.6, “Compression Options” for more details on compressed backups.

5.2.2 Restoring an Encrypted Backup Image

Restore an encrypted backup image named *<image_name>* to *<restoreDir>* on the server with `copy-back-and-apply-log`, using the encryption key contained in a file named *<keyFile>* :

Example 5.8 Restoring an Encrypted Backup Image

```
mysqlbackup --defaults-file=<my.cnf> --backup-image=<image_name> \
 --backup-dir=<backupTmpDir> --datadir=<restoreDir> --decrypt --key-file=<keyFile> copy-back-and-apply-log
```

See Section 14.13, “Encryption Options” for more details on backup encryption and decryption.

5.2.3 Restoring an Incremental Backup

There are different ways to use incremental backups to restore a database under different scenarios. The preferred method is to first restore the full backup and make it up-to-date to the time at which the full backup was performed using the `copy-back-and-apply-log` command (see Example 5.5, “Restoring a Backup Directory using `copy-back-and-apply-log`” or Example 5.6, “Restoring a Single-file Backup using `copy-back-and-apply-log`” on how to do it); then use `copy-back-and-apply-log` again to restore the incremental backup image on top of the full backup that was just restored:

Example 5.9 Restoring an Incremental Backup Image

```
mysqlbackup --defaults-file=<my.cnf> -uroot --backup-image=<inc_image_name> \
  --backup-dir=<incBackupTmpDir> --datadir=<restoreDir> --incremental \
  copy-back-and-apply-log
```

In this example, the incremental backup image named *<inc_image_name>* is restored to *<restoreDir>* on the server (where the full backup that the incremental backup image was based on has already been restored). The `--backup-dir` option is used to specify the temporary directory into which temporary output, status files, and backup metadata are saved. Repeat the step with other incremental backup images that you have, until the data has been restored to a desired point in time.

Alternatively you can bring your full backup up-to-date with your incremental backup. First, 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. You can keep updating it with more incremental backups, so it is ready to be restored anytime.

When an incremental backup is being restored using either the `copy-back-and-apply-log` or `apply-incremental-backup` command, the binary log (and also the relay log, in the case of a replica server), if included in the incremental backup, is also restored to the target server by default. This default behavior is overridden when either (1) the `--skip-binlog` option (or the `--skip-relaylog` option for the relay log) is used with the restore command, or (2) if the full backup the incremental backup was based on or any prior incremental backup that came in between the full backup and this incremental backup has the binary log (or relay log) missing.

See Section 4.3.2, “Making a Differential or Incremental Backup”, and Section 14.7, “Incremental Backup Options”, for more details on incremental backups.

5.2.4 Restoring Backups Created with the `--use-tts` Option

There are some special requirements for restoring backups created with transportable tablespaces (TTS) (that is, created with the `--use-tts` option):

- The destination server must be running.
- Make sure that the required parameters for connecting to the server (port number, socket name, etc.) are provided as command-line options for `mysqlbackup`, or are specified in the `[client]` section of a defaults file.
- The destination server must be using the same page size that was used on the server on which the backup was made.
- The innodb_file_per_table option must be enabled on the destination server.
- The tables being restored must not exist on the destination server.
- A restore fails if the InnoDB file format of a per-table data file (`.ibd` file) to be restored does not match the value of the `innodb_file_format` system variable on the destination server. In that case, use the `--force` option with the restore commands to change temporarily the value of `innodb_file_format` on the server, in order to allow restores of per-table data files regardless of their format.

When restoring a single-file backup created with the option setting `use-tts=with-minimum-locking`, the folder specified with `--backup-dir`, besides holding temporary output, status files, and metadata, is also used for extracting temporarily all the tables in the backup and for performing an `apply-log` operation to make the data up-to-date before restoring them to the server's data directory.

Selected tables can be restored from a backup created with transportable tablespaces (TTS) using the `--include-tables` and `--exclude-tables` options. The following command restores all tables in the “sales” database from the backup, but excludes the table with the name “hardware” :

Example 5.10 Restoring Selected Tables from a TTS Backup

```
mysqlbackup --socket=/tmp/restoreserver.sock --datadir=/logs/restoreserverdata --backup-dir=/logs/backup \
  --include-tables="^sales\." --exclude-tables="^sales\.hardware$" copy-back-and-apply-log
```

This following commands rename a table when restoring it from a TTS Backup by using the `--rename` option:

Example 5.11 Restoring and Renaming a Table from a TTS Backup

```
# Using fully qualified table names:
mysqlbackup --socket=/tmp/restoreserver.sock --datadir=/logs/restoreserverdata --backup-dir=/logs/backup \
  --include-tables="^sales\.cars" --rename="sales.cars to sales.autos" copy-back-and-apply-log

# It works the same if database names are omitted in the argument for --rename:
mysqlbackup --socket=/tmp/restoreserver.sock --datadir=/logs/restoreserverdata --backup-dir=/logs/backup \
  --include-tables="^sales\.cars" --rename="cars to autos" copy-back-and-apply-log
```

5.2.5 Restoring External InnoDB Tablespaces to Different Locations

When a backup contains external InnoDB tablespaces that resided outside of the backed-up server's data directory, you can restore them to locations different from their original ones by updating their path names in the .bl file inside the backup; see description of the file in Table 1.1, “Files in a MySQL Enterprise Backup Output Directory” for details.

5.2.6 Restoring a Backup from Cloud Storage to a MySQL Server

To restore a backup image from cloud storage to *`datadir`* on the server, use the cloud storage options, and also the `--backup-dir` option to specify the temporary directory into which temporary output, status files, and backup metadata will be saved:

Example 5.12 Restoring a Single-file Backup from an Oracle Cloud Infrastructure (OCI) Object Storage Classic Container to a MySQL Server

```
mysqlbackup \
--defaults-file=<my.cnf> \
--cloud-service=openstack  --cloud-container=<OCI Object Storage Classic container> \
--cloud-user-id=<serviceInstanceName>-<identityDomainName>:<userName> --cloud-password=<password> \
--cloud-tempauth-url=https://<dataCenterCode>.storage.oraclecloud.com \
--cloud-object=<backup_image_name> \
--datadir=/home/user/dba/datadir \
--backup-dir=/home/user/dba/orbackuptmpdir \
--backup-image=- \
copy-back-and-apply-log
```

Example 5.13 Restoring a Single-file Backup from an Oracle Cloud Infrastructure (OCI) Object Storage to a MySQL Server

```
mysqlbackup \
--defaults-file=<my.cnf> \
--cloud-service=openstack  --cloud-container=<OCI Object Storage bucket> \
-cloud-user-id=<OCI userName> --cloud-password='<OCI auth token>' \
--cloud-ca-info=/etc/ssl/certs/ca-certificates.crt
--cloud-basicauth-url=https://swiftobjectstorage.<region>.oraclecloud.com/v1/<OCI Object Storage namespace> \
--cloud-object=<backup_image_name> \
--datadir=<server_datadir> \
--backup-dir=/home/user/dba/orbackuptmpdir \
--backup-image=- \
copy-back-and-apply-log
```

Example 5.14 Restoring a Cloud Incremental Backup from an Oracle Cloud Infrastructure (OCI) Object Storage Service to a MySQL Server

```
mysqlbackup \--defaults-file=<my.cnf> \
--cloud-service=openstack  --cloud-container=<OCI Object Storage bucket> \
-cloud-user-id=<OCI userName> --cloud-password='<OCI auth token>' \
--cloud-ca-info=/etc/ssl/certs/ca-certificates.crt
--cloud-basicauth-url=https://swiftobjectstorage.<region>.oraclecloud.com/v1/<OCI Object Storage namespace> \
--cloud-object=<backup_image_name> \
--backup-image=- --datadir=<server_datadir> \
--backup-dir=/home/user/dba/orincrbackuptmpdir \
--incremental
```

```
copy-back-and-apply-log
```

Example 5.15 Restoring a Single-file Backup from an OpenStack Object Storage to a MySQL Server

```
mysqlbackup \
--defaults-file=<my.cnf> \
--cloud-service=openstack  --cloud-container=<swift container> \
--cloud-user-id=<keystone user> --cloud-password=<keystone password> \
--cloud-region=<keystone region> --cloud-tenant=<keystone tenant> \
--cloud-identity-url=<keystone url> --cloud-object=image_800.mbi \
--backup-dir=/home/user/dba/swiftbackuptmpdir \
--datadir=/home/user/dba/datadir \
--backup-image=- \
copy-back-and-apply-log
```

Example 5.16 Restoring a Single-file Backup from Amazon S3 to a MySQL Server

```
mysqlbackup\
--defaults-file=<my.cnf> \
--cloud-service=s3 --cloud-aws-region=<aws region> \
--cloud-access-key-id=<aws access key id> --cloud-secret-access-key=< aws secret access key> \
--cloud-bucket=<s3 bucket name> --cloud-object-key=<aws object key> \
--backup-dir=/home/user/dba/s3backuptmpdir \
--datadir=/home/user/dba/datadir \
--backup-image=- \
copy-back-and-apply-log
```

5.3 Point-in-Time Recovery

You can restore your database to its state at an arbitrary time using the binary log files included in the backups. The process assumes that following conditions are met:

- The backed-up MySQL Server has had its binary logging enabled. To check if this condition has been satisfied, perform this query on the server:

  ```
  mysql> SHOW VARIABLES LIKE 'log_bin';
  +---------------+-------+
  | Variable_name | Value |
  +---------------+-------+
  | log_bin       | ON    |
  +---------------+-------+
  1 row in set (0.00 sec)
  ```

  If the value of `log_bin` is `OFF`, binary logging has not been enabled. See The Binary Log on how to enable binary logging for the server.

- A series of backups, consisting typically of a full backup followed by a series of incremental backups, has been created for the server. The last backup in the series covers the targeted point in time for recovery. The example below illustrates such a typical case.

- The last backup in the backup series you have taken include in itself the relevant binary log files. (To ensure this requirement is satisfied, do not use any of the following MySQL Enterprise Backup options when creating the backup: `--skip-binlog`, `--use-tts`, `--no-locking`, or `--start-lsn`.)

These are the steps for a point-in-time recovery:

1. Restore the series of backups to the server, except for the last incremental backup in the series (which covers the targeted point in time for recovery). When finished, note the binary log position to which you have restored the server. The information is available from the `backup_variables.txt` file in the restored data directory of the server: look for the value of the entry `binlog_position` in the file. For example:

```
binlog_position=binlog.000012:426
```

This means after the restore of the backup series, the server is now at log position 426 found in the binary log file `binlog.000012`. You will need the information later.

Note

While the last binary log position recovered is also displayed by InnoDB after the restore, that is not a reliable means for obtaining the ending log position of your restore, as there could be DDL events and non-InnoDB changes that have taken place after the time reflected by the displayed position.

2. Extract the binary log from the last incremental backup in the backup series (that is, the backup that covers the targeted point in time for recovery). You do this by unpacking the incremental backup image into a backup directory using the `image-to-backup-dir` command; for example:

```
mysqlbackup --backup-dir=incr-backup-dir2 --backup-image=incremental_image2.bi image-to-backup-dir
```

Next, go into the resulting backup directory (`incr-backup-dir2` in this example) and, under the data directory inside, find the binary log file[s] (`binlog.000012` in this example):

```
incr-backup-dir2$ ls datadir
binlog.000012       ibbackup_logfile  mysql              pets      undo_002
...
```

3. Roll forward the database to its state at the targeted point in time for recovery, identified as t_R in this example, using the binary log file extracted in the last step. Then, using the `mysqlbinlog` utility, replay to the server the SQL activities recorded in the binary log file[s], from the log position the server has been restored to in Step 1 above (which is 426 in our example) all the way to time t_R. Specify the range of binary log events to replay using the `--start-position` option and the `--stop-position` option (which indicates the corresponding binary log position for t_R), and pipe the output to the `mysql` client:

```
mysqlbinlog --start-position="binary-log-position-at-the-end-of-backup-restores" \
         --stop-position="binary-log-position-corresponding-to-tR" \
         binary-log-filename  |   mysql -uadmin -p
```

Notes

- Using the `--start-datetime` or `--stop-datetime` option to specify the range of binary log segment to replay is not recommended: there is a higher risk of missing binary log events when using the option. Use `--start-position` and `--stop-position` instead.
- If you have more than one binary log files in your incremental backup and they are all needed for bringing the server up to its state at t_R, you need to pipe all of them to the server in a single connection; for example:

```
mysqlbinlog --start-position="426" --stop-position="binary-log-position-corresponding-t
  binlog.000012 binlog.000013 binlog.000014 |   mysql -u admin -p
```

You can also dump all the `mysqlbinlog` output to a single file first, and then pipe or play the file to the `mysql` client.

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

4. Check that the server has been restored to the desired point in time.

5.4 Restoring a Backup with a Database Upgrade or Downgrade

Important

You may encounter technical challenges during a server upgrade or downgrade, and it is beyond the function of MySQL Enterprise Backup, as a backup tool, to ensure a successful server upgrade or downgrade. Users interested in the topic are advised to consult the MySQL server manual, especially the Upgrading MySQL and Downgrading MySQL sections, and pay careful attention to the requirements and restrictions discussed there.

You can facilitate a server upgrade or downgrade by using MySQL Enterprise Backup to make a backup of your data from a *source server*, restore it to a new *target server*, and, after some preparations, start a different version of MySQL Server on the restored data. Here are a number of things that users should pay attention to when restoring a backup with a database upgrade or downgrade:

- *Restoring a database with a server downgrade* should only be performed when the MySQL servers on the source and the target servers are in the same release series. Downgrading to a lower series (for example, from 5.6.33 to 5.5.33) might cause server crashes or data corruption.
- *Restoring a database with a server upgrade* requires the following steps, the skipping of any of which might crash the restored server:

  1. Back up the data on the source server.
  2. Using *the same version of MySQL Enterprise Backup with which the backup was taken*, restore the data to the target server by running a `copy-back-and-apply-log` operation on the backup.
  3. Install on the target server *the same version of MySQL Server that was running on the source server when your backup was created*.
  4. Start the MySQL Server you just installed. Your restored data go through an abbreviated crash recovery process in preparation for a server upgrade.
  5. Perform a slow shutdown of the MySQL Server you just started in the last step by issuing the `SET GLOBAL innodb_fast_shutdown=0` statement and then shutting the server down. This ensures that all dirty pages are flushed, and hence there will be no redo log processing later for the upgraded server.
  6. Install the newer MySQL Server version on the target server.
  7. Start the newer MySQL Server version you just installed on the data directory you have restored and prepared in the earlier steps.
  8. Perform any other additional upgrade steps that might be required for your platform or distribution as documented in the MySQL reference manual. Make sure the mysql_upgrade that comes with your newer server version is applied.

  After performing these steps, check your data to make sure that your restore has been successful.

Chapter 6 Using MySQL Enterprise Backup with Replication

Table of Contents

6.1 Setting Up a New Replica 67
6.2 Backing up and Restoring a Replica Database 69
6.3 Restoring a Source Database 70

Backup and restore operations are especially important in systems that use MySQL replication to synchronize data across a source server and a set of replica servers. In a replication configuration, MySQL Enterprise Backup helps you manage images for the entire system, set up new replica servers, or restore a source server in an efficient way that avoids unnecessary work for the replica servers. On the other hand, having multiple replica 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 the database to a specific point in time, even a time that is later than that of the last backup.

6.1 Setting Up a New Replica

MySQL Enterprise Backup allows you to set up a replica by backing up the source and restoring the backup on a new replica server, without having to stop the source.

For servers NOT using GTID:

1. Take a full backup of the source and then use, for example, the `copy-back-and-apply-log` command, to restore the backup and the log files to the right directories on the new replica and prepare the data.

   Note

   Do not use the `--no-locking` option when backing up the server, or you will be unable to get a proper binary log position in Step 4 below for initializing the replica.

2. Edit the `my.cnf` file of the new replica and put `skip-slave-start` and `event_scheduler=off` (if the source uses the Event Scheduler) under the `[mysqld]` section.

3. Start the new replica `mysqld` (version >= 5.1). You see the following in the server's output:

   ```
   …
   InnoDB: Last MySQL binlog file position 0 128760007, file name ./hundin-bin.000006
   …
   ```

   While a `Last MySQL binlog file position` has been displayed, it is NOT necessarily the latest binary log position on the backed up server, as InnoDB does not store binary log position information for any DDL operations or any changes to non-InnoDB tables. *Do not use this binary log position to initialize the replica*. The next step explains how to find the correct binary log position to use.

4. Look for the file *`datadir`*`/meta/backup_variables.txt` where *`datadir`* is the data directory of the new replica. Look into the file to retrieve the latest binary log position and the corresponding log file number stored inside:

   ```
   binlog_position=hundin-bin.000006:128760128
   ```

5. Use the `CHANGE MASTER TO` SQL statement and the information you have retrieved in the last step to initialize the replica properly:

```
CHANGE MASTER TO
MASTER_LOG_FILE='hundin-bin.000006',
MASTER_LOG_POS=128760128;
```

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

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

7. Remove the line `skip-slave-start` and `event_scheduler=off` entries you added to the `my.cnf` file of the replica in step 2. (You can also leave the `skip-slave-start` entry in, but then you will always need to use the START SLAVE statement to start replication whenever you restart the replica server.)

8. Restart the replica server. Replication starts.

For servers using GTIDs (supported by MySQL Server 5.6 and after; see Setting Up Replication Using GTIDs on how to enable servers to use GTIDs):

1. Take a full backup of the source and then use, for example, the `copy-back-and-apply-log` command, to restore the backup and the log files to the right directories on a new GTID-enabled replica and prepare the data.

2. Edit the `my.cnf` file of the new replica and put `skip-slave-start` and `event_scheduler=off` (if the source uses the Event Scheduler) under the `[mysqld]` section.

3. Start the new replica server.

4. Connect to the replica server with the `mysql` client. Then, execute the following statement to reset the binary log:

   ```
   mysql> RESET MASTER;
   ```

   And execute the following statement to stop the binary logging:

   ```
   mysql> SET sql_log_bin=0;
   ```

5. When a server using the GTID feature is backed up, `mysqlbackup` produces a file named `backup_gtid_executed.sql`, which can be found in the restored data directory of the new replica server. The file contains a SQL statement that sets the `GTID_PURGED` configuration option on the replica:

   ```
   # On a new replica, issue the following command if GTIDs are enabled:
   SET @@GLOBAL.GTID_PURGED='f65db8e2-0e1a-11e5-a980-080027755380:1-3';
   ```

   It also contains a commented-out `CHANGE MASTER TO` statement for initializing the replica:

   ```
   # Use the following command if you want to use the GTID handshake protocol:
   # CHANGE MASTER TO MASTER_AUTO_POSITION = 1;
   ```

   Uncomment the command and add any needed connection and authentication parameters to it (for example, `MASTER_HOST`, `MASTER_USER`, `MASTER_PASSWORD`, and `MASTER_PORT`):

   ```
   # Use the following command if you want to use the GTID handshake protocol:
   CHANGE MASTER TO MASTER_HOST='127.0.0.1', MASTER_USER='muser', MASTER_PASSWORD='mpass', MASTER_PORT=18675,
   ```

   Execute the file with the `mysql` client

   ```
   mysql> source /path-to-backup_gtid_executed.sql/backup_gtid_executed.sql
   ```

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

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

7. Remove the `skip-slave-start` and `event_scheduler=off` entries you added to the `my.cnf` file of the replica in step 2. (You can also leave the `skip-slave-start` entry in, but then you will always need to use the START SLAVE statement to start replication whenever you restart the replica server.)

8. Restart the replica server. Replication starts.

For more information on the GTIDs, see GTID feature.

6.2 Backing up and Restoring a Replica Database

To backup a replica database, add the `--slave-info` option to your backup command.

To restore the backup on a replica server, follow the same steps outlined in Section 6.1, “Setting Up a New Replica”.

Temporary tables on statement-based replication (SBR) replica. MySQL Enterprise Backup does not include temporary tables inside a backup. As a result, for a replica server in a statement-based replication (SBR) or a mixed-based replication setup (see Replication Formats for details), any temporary tables still open at the end of the backup process will be missing in the restored replica server, making the replication state of the replica inconsistent, and any subsequent replicated statements that refer to the temporary tables will fail. To avoid the issue, after the hot backup phase in which `mysqlbackup` copies all the InnoDB tables, it enters into a loop, in which the following happens:

1. `mysqlbackup` waits until all temporary tables have been closed by the replication SQL thread. `mysqlbackup` tells if that is the case by checking if the variable `Slave_open_temp_tables` has a zero value.

2. After `Slave_open_temp_tables=0` is detected, `mysqlbackup` stops the replication SQL thread to prevent more changes to the tables on the replica.

3. To avoid the unexpected consequence by a race condition, after the replication SQL thread has been stopped, `mysqlbackup` checks once more if `Slave_open_temp_tables=0` is still true:

   - If it is true, `mysqlbackup` exits the loop and finishes the backup by asserting a global read lock and copies all the non-InnoDB tables.

   - If it is not true, new temporary tables have just been created and opened on the replica. `mysqlbackup` then restarts the replication SQL thread, so more updates can be made on the replica servers. `mysqlbackup` then goes back to step 1 of this loop

Besides the exit condition described in step (3) above (which is, there really are no more open temporary tables and `mysqlbackup` is ready to complete the backup), `mysqlbackup` will time out after staying in the above loop for too long to wait for all temporary tables to be closed. The duration `mysqlbackup` waits until it times out is specified by the `--safe-slave-backup-timeout` option.

In addition, `mysqlbackup` also runs an initial check at the beginning of a replica backup to see if `Slave_open_temp_tables=0` becomes true within the duration set by `--safe-slave-backup-timeout`. See description for `--safe-slave-backup-timeout` on details about the check.

Notice that the above-described issue with temporary tables does not exist for a row-based replication (RBR) setup, for which temporary tables are not replicated onto the replica. User who are certain that SBR is not occurring for the replica can set `--safe-slave-backup-timeout=0`, which will prevent `mysqlbackup` from entering the above-mentioned loop.

Note

See the limitation that applies when backing up a replica in Appendix B, *Limitations of MySQL Enterprise Backup*.

6.3 Restoring a Source Database

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

1. Shut down the source database and then use, for example, the `copy-back-and-apply-log` command, to restore a backup of it and prepare the data.
2. Edit the source `my.cnf` file and comment out `log-bin`, so that the replicas do not receive twice the binary log needed to recover the source.
3. Replication in the replicas must be stopped temporarily while you pipe the binary log to the source. In the replicas, do:

   ```
   mysql> STOP SLAVE;
   ```

4. Start the source `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.000002
   …
   ```

   InnoDB prints the binary log file (`./omnibook-bin.000002` in this case) and the position (`5585832` in this case) it was able to recover to.

5. Pipe the remaining of the binary log files to the restored server. The number of remaining binary log files varies depending on the length of the timespan between the last backup and the time to which you want to bring the database up to date. The longer the timespan, the more remaining binary log files there may be. All the binary log files, containing all the continuous binary log positions in that timespan, are required for a successful restore.

   You also need to supply the starting position in the binary log by which the piping of the events should start. Deduce that information from the `meta/backup_variables.txt` file in the backup you just restored in step 1 above (access `backup_variables.txt` by, for example, going to the temporary backup directory you specified with `--backup-dir` during the restore, and find the file under the `meta` folder): look for the entry `binlog_position=`**`value`** in `meta/backup_variables.txt`, and supply **`value`** to `mysqlbinlog` with the `--start-position` option.

   Note

   While the last binary log position recovered is also displayed by InnoDB after the restore (see step 4 above), that is not a reliable number for deducing the start position for `mysqlbinlog` to use, as there could be DDL events and non-InnoDB changes that have taken place after the time reflected by the displayed position.

   For example, if there are two more binary log files, `omnibook-bin.000003` and `omnibook-bin.000004` that come after `omnibook-bin.000002` and the recovery in step 4 above has ended by `5585834` *according to the* `backup_variables.txt` *file*, pipe the binary log with a single connection to the server with this command:

```
$ mysqlbinlog --start-position=5585834 /mysqldatadir/omnibook-bin.000002 \
  /mysqldatadir/omnibook-bin.000003 /mysqldatadir/omnibook-bin.000004 | mysql
```

See Point-in-Time (Incremental) Recovery Using the Binary Log for more instructions on using `mysqlbinlog`.

6. The source database is now recovered. Shut down the source and edit `my.cnf` to uncomment `log-bin`.
7. Start the source again.
8. Start replication in the replicas again:

```
mysql> START SLAVE;
```

Chapter 7 Performance Considerations for MySQL Enterprise Backup

Table of Contents

7.1 Optimizing Backup Performance 73
7.2 Optimizing Restore Performance 76

This chapter describes the performance considerations for backing up and restoring databases using MySQL Enterprise Backup.

7.1 Optimizing Backup Performance

This section describes the performance considerations for backing up a database with MySQL Enterprise Backup. 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 14.7, “Incremental Backup Options” for information about these options. For usage instructions for the backup and apply stages of incremental backups, see Section 4.3.2, “Making a Differential or 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 4.3.3, “Making a Compressed Backup” and Section 14.6, “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 13.5, “Single-File Backup Operations” for the options related to single-file backups, and Section 4.3.5, “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, as explained in Optimizing InnoDB Configuration Variables. 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

`mysqlbackup` can take advantage of modern multicore CPUs and operating system threads to perform backup operations in parallel. See Section 14.10, “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 once the system `iowait` value reaches approximately 20%.

MyISAM Considerations

Important

- Although `mysqlbackup` 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 `mysqlbackup` returned exit code 0, or by observing that `mysqlbackup` has printed the text “mysqlbackup completed OK!”.
- `mysqlbackup` 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 Appendix B, *Limitations of MySQL Enterprise Backup* 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 14.8, “Partial Backup and Restore Options” for information about the relevant options, and Section 4.3.4, “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 4.3.4, “Making a Partial Backup”.
- It allows the use of table compression for InnoDB tables.

In general, using table compression by having `ROW_FORMAT=COMPRESSED` decreases table sizes and increase backup and restore performance. However, as a trade-off, table compression can potentially increase redo log sizes and thus slow down incremental backups and restores, as well as `apply-log` operations. See How Compression Works for InnoDB Tables for details.

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 13.2, “Apply-Log Operations”

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 13.2, “Apply-Log Operations”.

7.2 Optimizing Restore Performance

This section describes the performance considerations for restoring a database with MySQL Enterprise Backup. This subject is important 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 13.3, “Restore Operations”.

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 14.8, “Partial Backup and Restore 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 14.6, “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 13.5, “Single-File Backup Operations” for information about making single-file backups.

The Apply-Log Phase

See The Apply-Log Phase for performance considerations regarding the apply-log phase.

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

`mysqlbackup` can take advantage of modern multicore CPUs and operating system threads to perform backup operations in parallel. See Section 14.10, “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 Encryption for Backups

In order to enhance security for backed up data, MySQL Enterprise Backup provides encryption for single-file backups. The encryption can also be applied when creating a partial, compressed, or incremental single-file backups, and for streaming backup data to another device or server.

The encryption is performed with Advanced Encryption Standard (AES) block cipher in CBC mode, with a key string of 64 hexadecimal digits supplied by the user. Decryption is performed using the same key. The key can be created manually just by putting together 64 random hexadecimal bytes, or it can be generated by `shasum` (or similar programs for hash calculations that work on your platform) by supplying it with a keyphrase:

```
$ echo -n "my secret passphrase" | shasum -a 256
a7e845b0854294da9aa743b807cb67b19647c1195ea8120369f3d12c70468f29  -
```

Note that the "-" at the end is not part of the key and should be ignored. Supply the key to `mysqlbackup` with the `--key` option, or paste the key into a key file and supply the file's pathname to `mysqlbackup` with the `--key-file` option.

To generate a key randomly, you can use tools like OpenSSL:

```
$ openssl rand -hex 32
8f3ca9b850ec6366f4a54feba99f2dc42fa79577158911fe8cd641ffff1e63d6
```

To put an OpenSSL-generated key into a key file, you can do the following:

```
$ openssl rand -hex 32 >keyfile
$ cat keyfile
6a1d325e6ef0577f3400b7cd624ae574f5186d0da2eeb946895de418297ed75b
```

The encryption function uses MySQL Enterprise Backup's own encryption format, which means decryption is possible only by using MySQL Enterprise Backup. For Unix-like operating systems, different magic numbers are used to identify encrypted and unencrypted backup files. For example, you can add these lines to the `/etc/magic` file of your operating system:

```
0   string  MBackuP\n   MySQL Enterprise Backup backup image
0   string  MebEncR\n   MySQL Enterprise Backup encrypted backup
```

The `file` command can then be used to identify the file types:

```
  $ file /backups/image1 /backups/image2
  /backups/image1: MySQL Enterprise Backup backup image
  /backups/image2: MySQL Enterprise Backup encrypted backup
```

The command options used for encryption and decryption are `--encrypt`, `--decrypt`, `--key`, and `--key-file`. These options can be used with various operations on backup images. See Section 14.13, "Encryption Options" for details.

The following is a sample command for creating an encrypted backup:

```
mysqlbackup --backup-image=/backups/image.enc --encrypt
   --key=23D987F3A047B475C900127148F9E039485798364519287 4A2B3049570C12A34
   --backup-dir=/var/tmp/backup  backup-to-image
```

To use a key file for the same task:

```
mysqlbackup --backup-image=/backups/image.enc --encrypt
   --key-file=/meb/key  --backup-dir=/var/tmp/backup  backup-to-image
```

To decrypt a backup when extracting it:

```
mysqlbackup --backup-image=/backups/image.enc --decrypt
   --key-file=/meb/key --backup-dir=/backups/extract-dir  extract
```

To validate an encrypted backup image:

```
mysqlbackup --backup-image=/logs/encimage.bi --decrypt --key-file=/meb/enckey validate
```

Chapter 9 Using MySQL Enterprise Backup with Media Management Software (MMS) Products

Table of Contents

9.1 Backing Up to Tape with Oracle Secure Backup 81

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.

9.1 Backing Up to Tape with Oracle Secure Backup

Tape drives are affordable, high-capacity storage devices for backup data. MySQL Enterprise Backup 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 `mysqlbackup` to control various aspects of the SBT processing. The `--sbt-*` options are listed in Section 14.9, “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 `mysqlbackup` 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 `mysqlbackup` 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.

A backup to tape always uses one write thread.

To restore MySQL data from tape:

- Specify the `--backup-image=sbt:`*`name`* parameter of `mysqlbackup` 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 9.1 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 10 Monitoring Backups with MySQL Enterprise Monitor

The MySQL Enterprise Monitor is a companion product to the MySQL Server that enables monitoring of MySQL instances and their hosts, notification of potential issues and problems, and advice on how to correct issues. Among its other functions, it can be used to monitor the progress and history of backup jobs. Check the MySQL Enterprise Monitor User's Guide for detail.

Chapter 11 Troubleshooting for MySQL Enterprise Backup

Table of Contents

11.1 Error codes of MySQL Enterprise Backup 85
11.2 Working Around Corruption Problems 85
11.3 Using the MySQL Enterprise Backup Logs 86
11.4 Using the MySQL Enterprise Backup Manifest 88

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 B, *Limitations of MySQL Enterprise Backup*.
- If `mysqlbackup` 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 `mysqlbackup` is sent to `stderr` rather than `stdout`. By default, the same output is also saved to a log file in the `backup_dir` for use in error diagnosis. See Section 14.11, “Message Logging Options” for details on how to configure this logging feature.
- 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 `mysqlbackup` 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 those tables to monitor ongoing backup jobs, see how much time has been used for various stages, and check if any errors have occurred.

11.1 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.

11.2 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
mysqlbackup: Re-reading page at offset 0 3185082368 in /sqldata/mts/ibdata15
mysqlbackup: 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.

11.3 Using the MySQL Enterprise Backup Logs

Besides the message output of MySQL Enterprise Backup to the `stderr` stream and the log file, progress and history of each backup are also logged into the `mysql.backup_progress` and `mysql.backup_history` tables on the backed-up servers (to skip updating the two tables, use the `--no-history-logging` option with the backup command).

`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:

```
mysql> DESCRIBE mysql.backup_progress;
+---------------+---------------+------+-----+-------------------+-----------------------------------------
| Field         | Type          | Null | Key | Default           | Extra
+---------------+---------------+------+-----+-------------------+-----------------------------------------
| backup_id     | bigint(20)    | NO   |     | NULL              |
| tool_name     | varchar(4096) | NO   |     | NULL              |
| error_code    | int(11)       | NO   |     | NULL              |
| error_message | varchar(4096) | NO   |     | NULL              |
| current_time  | timestamp     | NO   |     | CURRENT_TIMESTAMP | DEFAULT_GENERATED on update CURRENT_TIMESTA
| current_state | varchar(200)  | NO   |     | NULL              |
+---------------+---------------+------+-----+-------------------+-----------------------------------------
6 rows in set (0.00 sec)
```

The `backup_progress` table is in CSV format. You can query the table with the `mysql` client, or parse the corresponding `.CSV` file with an application or script.

Here are some ways to make use of the information in the `backup_progress` table:

- Use the `backup_id` value to query all the information for different stages of a single backup operation, and to find the corresponding row in the `backup_history` table for the same backup (the row is written to the `backup_history` table only after the backup is finished).

- Use the `error_code` and `error_message` values to track any errors that have occurred, and to see if the backup operation should be terminated because of any serious errors.

- Use the current_time and current_state values to track the progress of the operation. They also allow you to measure how long each stage of the backup takes, which helps you plan for your future backups.

backup_history Table

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

```
mysql> DESCRIBE mysql.backup_history;
+--------------------------+---------------+------+-----+---------------------+-------+
| Field                    | Type          | Null | Key | Default             | Extra |
+--------------------------+---------------+------+-----+---------------------+-------+
| backup_id                | bigint(20)    | NO   |     | NULL                |       |
| tool_name                | varchar(4096) | NO   |     | NULL                |       |
| start_time               | timestamp     | NO   |     | 0000-00-00 00:00:00 |       |
| end_time                 | timestamp     | NO   |     | 0000-00-00 00:00:00 |       |
| binlog_pos               | bigint(20)    | NO   |     | NULL                |       |
| binlog_file              | varchar(255)  | NO   |     | NULL                |       |
| compression_level        | int(11)       | NO   |     | NULL                |       |
| engines                  | varchar(100)  | NO   |     | NULL                |       |
| innodb_data_file_path    | varchar(2048) | NO   |     | NULL                |       |
| innodb_file_format       | varchar(100)  | NO   |     | NULL                |       |
| start_lsn                | bigint(20)    | NO   |     | NULL                |       |
| end_lsn                  | bigint(20)    | NO   |     | NULL                |       |
| incremental_base_lsn     | bigint(20)    | NO   |     | NULL                |       |
| backup_type              | varchar(50)   | NO   |     | NULL                |       |
| backup_format            | varchar(50)   | NO   |     | NULL                |       |
| mysql_data_dir           | varchar(2048) | NO   |     | NULL                |       |
| innodb_data_home_dir     | varchar(2048) | NO   |     | NULL                |       |
| innodb_log_group_home_dir | varchar(2048) | NO   |     | NULL                |       |
| innodb_log_files_in_group | varchar(100)  | NO   |     | NULL                |       |
| innodb_log_file_size     | varchar(100)  | NO   |     | NULL                |       |
| backup_destination       | varchar(4096) | NO   |     | NULL                |       |
| lock_time                | double(7,3)   | NO   |     | NULL                |       |
| exit_state               | varchar(10)   | NO   |     | NULL                |       |
| last_error               | varchar(4096) | NO   |     | NULL                |       |
| last_error_code          | int(11)       | NO   |     | NULL                |       |
| start_time_utc           | bigint(20)    | NO   |     | NULL                |       |
| end_time_utc             | bigint(20)    | NO   |     | NULL                |       |
| consistency_time_utc     | bigint(20)    | NO   |     | NULL                |       |
| meb_version              | varchar(20)   | NO   |     | 0.0.0               |       |
+--------------------------+---------------+------+-----+---------------------+-------+
29 rows in set (0.00 sec)
```

Warning

Because a successful backup is always recorded as such in the backup_history table, a failure in the apply-log phase of a backup-and-apply-log command is not reflected in the backup_history table. It is always important to check the output of mysqlbackup to see if an operation is completed fully without an error.

Here is information on some columns of the backup_history table, and some ways to make use of the information:

- The tool_name column records the full mysqlbackup command that triggers the backup, including all the options used.
- You can use the end_lsn value of your latest backup as the starting LSN value for you next incremental backup by specifying it with the --start-lsn option. (An alternative to specifying the start LSN value for an incremental backup is to use the --incremental-base option).

- The `binlog_pos` column gives the position of the binary log up to where log events have been covered by the backup. Because the `backup_history` table used to be in the CSV format, which cannot register `NULL` values directly, if binary logging is not enabled, a value of `-1` is entered into the column; the same applies to other columns for the logging of `NULL` values.
- The value for `backup_type` is one of `FULL`, `PARTIAL`, `INCREMENTAL`, or `TTS`.
- The value for `backup_format` is one of `IMAGE` (for single-file backups) or `DIRECTORY` (for directory backups).
- Use the values that show the backup's 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.
- The value for `exit_state` is either `SUCCESS` or `FAILURE`. If the `exit_state` is `SUCCESS` and `last_error` is `'NO_ERROR'`, the backup operation has been successful; when it is not the case, see `last_error` and `last_error_code` for the latest error of the operation. To retrieve the full list of errors for that backup operation, go to the `backup_progress` table.

11.4 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 involved 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. 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. The file also contains details of all the plugins defined on the backed-up server, by which users should make sure the same plugins are defined in the same manner on the target server for restoration.
- `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` commands.) 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` commands, or to remap the paths of extracted files with the `--src-entry` and `--dst-entry` options.

Part III `mysqlbackup` Command Reference

Table of Contents

12 mysqlbackup 93
13 mysqlbackup commands 95
13.1 Backup Operations 95
13.2 Apply-Log Operations 96
13.3 Restore Operations 97
13.4 Validation Operations 99
13.5 Single-File Backup Operations 101
14 mysqlbackup Command-Line Options 105
14.1 Standard Options 112
14.2 Connection Options 114
14.3 Server Repository Options 116
14.4 Backup Repository Options 118
14.5 Metadata Options 122
14.6 Compression Options 123
14.7 Incremental Backup Options 124
14.8 Partial Backup and Restore Options 127
14.9 Single-File Backup Options 133
14.10 Performance / Scalability / Capacity Options 136
14.11 Message Logging Options 143
14.12 Progress Report Options 144
14.13 Encryption Options 148
14.14 Cloud Storage Options 148
14.15 Options for Special Backup Types 151
15 Configuration Files and Parameters 155

Chapter 12 mysqlbackup

The mysqlbackup client 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 with 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.
- Any other files in the database subdirectories under the server's data directory.

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, or to other places.

Here are some sample commands to start a backup operation with 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 the [mysqlbackup] section, 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 4.1.2, “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 Chapter 14, *mysqlbackup Command-Line Options*. For information about configuration files, see Chapter 15, *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 `mysqlbackup` 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 `mysqlbackup` 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` 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 the `mysqlbackup` command has been completed successfully by verifying that `mysqlbackup` has returned the exit code 0, or by observing that `mysqlbackup` has printed the text “mysqlbackup completed OK!”.

- `mysqlbackup` 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 Appendix B, *Limitations of MySQL Enterprise Backup* for restrictions on creating backups in parallel with the DDL operations.

Chapter 13 `mysqlbackup` commands

Table of Contents

13.1 Backup Operations 95
13.2 Apply-Log Operations 96
13.3 Restore Operations 97
13.4 Validation Operations 99
13.5 Single-File Backup Operations 101

These are commands for the major operations for `mysqlbackup`. Only one of them can be specified for each `mysqlbackup` invocation, and, unlike the command options, the name of a command is not preceded by any dashes.

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

The major groups of commands are:

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

13.1 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` commands 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-RESTORE-OPTIONS]
            [PERFORMANCE-SCALABILITY-CAPACITY-OPTIONS]
            [MESSAGE-LOGGING-OPTIONS]
            [PROGRESS-REPORT-OPTIONS]
            backup | backup-and-apply-log

mysqlbackup [STD-OPTIONS]
            [CONNECTION-OPTIONS]
            [SERVER-REPOSITORY-OPTIONS]
```

```
            [BACKUP-REPOSITORY-OPTIONS]
            [METADATA-OPTIONS]
            [COMPRESSION-OPTIONS]
            [SPECIAL-BACKUP-TYPES-OPTIONS]
            [INCREMENTAL-BACKUP-OPTIONS]
            [PARTIAL-BACKUP-RESTORE-OPTIONS]
            [SINGLE-FILE-BACKUP-OPTIONS]
            [PERFORMANCE-SCALABILITY-CAPACITY-OPTIONS]
            [MESSAGE-LOGGING-OPTIONS]
            [PROGRESS-REPORT-OPTIONS]
            [ENCRYPTION-OPTIONS]
            [CLOUD-STORAGE-OPTIONS]
            backup-to-image
```

- `backup`

  Backs up data to a directory. In most cases, single-file backups, which are created using the `backup-to-image` command, are preferred over directory backups.

  The command only performs the initial phase of a complete backup process. The second phase is performed later by running `mysqlbackup` again with the `apply-log` command, which makes the backup consistent.

- `backup-and-apply-log`

  A combination of `backup` and `apply-log`. It cannot be used for an incremental backup.

  Note

  For the `backup-and-apply-log` command, the compression options are only supported for MySQL Enterprise Backup 3.12.3 and later.

- `backup-to-image`

  Produces a single-file backup holding the backup data. In most cases, single-file backups are preferred over directory backups, which are created using the `backup` command.

  The command 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 stream.

  The command also requires the use of the `--backup-dir` option to supply a temporary folder to save the backup metadata (including the `mysqlbackup` message log, the start and end LSN, and so on) and some temporary output. Note that, however, except when streaming the backup image with `--backup-image=-`, if `--backup-image` does not give a full path name, `mysqlbackup` will actually take the value of `--backup-image` as a path relative to the directory specified by `--backup-dir`, and thus store the single-file backup under `--backup-dir` (or, if the `--with-timestamp` option is used, under a subdirectory created under `--backup-dir` that bears the timestamp in its name).

13.2 Apply-Log Operations

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` command, you must run the steps separately when performing incremental backups.

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

mysqlbackup [STD-OPTIONS]
            [--incremental-backup-dir=PATH] [--backup-dir=PATH]
            [--limit-memory=MB] [--uncompress]
            [MESSAGE-LOGGING-OPTIONS]
            [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 13.1 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` defaults 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.

You can also perform `apply-log` and `copy-back` (which restores the prepared backup) together with a single `copy-back-and-apply-log` command.

13.3 Restore Operations

The restore operations restores the data files from a backup to their original locations on the database server, or to other desired locations. The MySQL instance must be shut down first before a restore operation (except for backups created using transportable tablespace (TTS)). 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 examples, see Chapter 5, *Recovering or Restoring a Database*.

```
mysqlbackup [STD-OPTIONS]
            [SERVER-REPOSITORY-OPTIONS]
            [--backup-dir=PATH]
            [--uncompress]
            [MESSAGE-LOGGING-OPTIONS]
            [PARTIAL-BACKUP-RESTORE-OPTIONS]
            [PROGRESS-REPORT-OPTIONS]
            [CLOUD-STORAGE-OPTIONS]
            copy-back

mysqlbackup [STD-OPTIONS]
            [SERVER-REPOSITORY-OPTIONS]
            [--backup-image=IMAGE]
            [--backup-dir=PATH]
            [--uncompress]
```

```
            [MESSAGE-LOGGING-OPTIONS]
            [PARTIAL-BACKUP-RESTORE-OPTIONS]
            [PROGRESS-REPORT-OPTIONS]
            [ENCRYPTION-OPTIONS]
            [CLOUD-STORAGE-OPTIONS]
            copy-back-and-apply-log
```

- `copy-back`

  Restores files from a directory backup to their original locations within the MySQL server.

  Before restoring a hot backup using the `copy-back` command, the backup has to be prepared and made consistent using the `apply-log` command. See Section 5.1, “Preparing the Backup to be Restored” for details. You can also perform `apply-log` and `copy-back` together with a single `copy-back-and-apply-log` command.

  Some clean-up efforts on the target directory for restoration might be needed before performing a full restore (for example, when the backup data is used to set up a new MySQL server or to replace all data of an existing MySQL server). See Section 5.2, “Performing a Restore Operation” [59] for details.

  There are some special requirements when restoring backups created with the `--use-tts` option; see Section 5.2.4, “Restoring Backups Created with the `--use-tts` Option” for details.

- `copy-back-and-apply-log`

  In a single step, restores a single-file backup specified by the `--backup-image` option or a backup from the directory specified by the `--backup-dir` option to a server's data directory and performs an `apply-log` operation to the restored data to bring them up-to-date. Comparing with a multi-step approach for restoring a single-file backup (which typically consists of performing the successive steps of extract, uncompress,apply-log, and copy-back for restoring compressed image, or extract ,apply-log, and copy-back for uncompressed image), the command makes the restoration process simpler and faster, and also saves the disk space required.

  The following are some special requirements for different kinds of backup restoration using `copy-back-and-apply-log`:

  - To restore a compressed directory or image, include the `--uncompress` option in the command line.
  - To restore a single-file backup, besides specifying the location of the backup image with the `--backup-image` option, also supply with the `--backup-dir` option the location of a folder that will be used for storing temporary files produced during the restoration process.
  - To restore an incremental backup directory, assuming the full backup (on which the incremental backup was based) has already been restored:
    - Include the `--incremental` option in the command line.
    - Use either the `--backup-dir` or `--incremental-backup-dir` option to specify the incremental backup directory.
  - To restore a single-file incremental backup, besides specifying the location of the incremental backup image with the `--backup-image` option, also supply with the `--backup-dir` option the location of a folder that will be used for storing temporary files produced during the restoration process.
  - To restore a backup created with the `--use-tts option`:
    - See the general requirements described in Section 5.2.4, “Restoring Backups Created with the `--use-tts` Option”.

- When restoring a single-file backup created with the option setting `use-tts`=`with-minimum-locking`, the folder specified with `--backup-dir` is also used for extracting temporarily all the tables in the backup and for performing an `apply-log` operation to make the data up-to-date before restoring them to the server's data directory.
- When restoring a backup directory created with the option `--use-tts`=`with-minimum-locking`, an `apply-log` operation will be performed on the backup directory. That means the backup taken will be altered during the process, and users might want to make an extra copy of the backup directory before proceeding with the restoration, in order to prevent the loss of backup data in case something goes wrong.

Also note that:

- Backups created with the `--skip-unused-pages` option cannot be restored using `copy-back-and-apply-log`.
- For image backups taken with MySQL Enterprise Backup 3.8.2 or earlier, per-table `.ibd` files pointed to by `.isl` files in a backup are restored by `copy-back-and-apply-log` to the server's data directory rather than the locations pointed to by the .isl files.
- Due to a known issue, when restoring a compressed backup created with MySQL Enterprise Backup 3.9 or earlier and containing any InnoDB tables that were created on the server as compressed tables (by using the `ROW_FORMAT=COMPRESSED` option, the `KEY_BLOCK_SIZE=` option, or both), do not use `copy-back-and-apply-log`; instead, perform an `apply-log` first, and then a `copy-back`. See entry for Bug# 17992297 in the MySQL Enterprise Backup 3.10.0 changelog for details.

At the end of the `copy-back-and-apply-log` operation, the file `backup_variables.txt` is being created or updated in the data directory. This file contains metadata about the restored contents and is being used by successive single-step restores of incremental backups; it should not be deleted or modified by users.

For some sample commands for restoring different kinds of backups with the `copy-back-and-apply-log` command, see Section 5.2, “Performing a Restore Operation”.

Warning

When restoring a server for replication purpose, if the backed-up server has used the `innodb_undo_directory` option to put the undo logs outside of the data directory, when using the file `server-my.cnf` or `server-all.cnf` for the `--defaults-file` option with `copy-back` or `copy-back-and-apply-log`, care should be taken to configure correctly the `innodb_undo_directory` option in the file. Otherwise, the data or log files on the original server might be overwritten by accident.

13.4 Validation Operations

To ensure the integrity of the backup data, MySQL Enterprise Backup provides a `validate` command for validating a backup by the checksum values of its data pages after the backup is created or transferred to another system.

```
mysqlbackup [STD-OPTIONS]
             [--backup-dir=PATH][--backup-image=IMAGE]
            [MESSAGE-LOGGING-OPTIONS]
            [PROGRESS-REPORT-OPTIONS]
            [CLOUD-STORAGE-OPTIONS]
            validate
```

- `validate`

  Verifies that a backup is not corrupted, truncated, or damaged. This operation validates the checksum value for each data page in a backup.

  To avoid spending excessive time and resources on files that are too heavily corrupted, `mysqlbackup` stops validating a .ibd file after more than twenty corrupted pages are found in it, and proceeds to the next file instead. In that case, the operation's summary will not give a full count of corrupted pages, but only says “at least 20 pages are corrupted.”

  The operation also has the following limitations:

  - For any backup directory, the operation can only validate the InnoDB data files (`ibdata*` and `*.ibd` files) in it. Problems with other file types within a backup directory (for example, `.frm` file corruptions) are not detected.
  - If any `.ibd` or `.frm` files are missing from a backup directory during a backup or have been deleted from a backup directory after the backup was made, the `validate` operation will not be able to detect the problem.
  - If a backup directory has been corrupted by removing or truncating pages from any of the .ibd files inside , the `validate` operation will not be able to detect the problem.

  Here is a sample command for validating a backup directory:

```
mysqlbackup -uroot --backup-dir=/logs/backupext validate
```

  Here is a sample command for validating a backup image:

```
mysqlbackup -uroot --backup-image=/logs/fullimage.mi  validate
```

  The following is a sample command for validating an encrypted backup image and the output for the successful validation:

```
$ mysqlbackup –backup-image=/meb/backups/image.mbi --decrypt --key-file=/meb/enckeyfile validate

140219 11:22:44 mysqlbackup: INFO: Validating image ... /logs/img.bi
140219 11:22:44 mysqlbackup: INFO: Validate: [Dir]: meta
140219 11:22:45 mysqlbackup: INFO: Total files as specified in image: 44
mysqlbackup: INFO: datadir/tpch/tabnorm7.ibd Validated...
mysqlbackup: INFO: datadir/tpch/tabnorm8.ibd Validated...
mysqlbackup: INFO: datadir/tpch/tabnorm9.ibd Validated...
................
140219 11:22:45 mysqlbackup: INFO: Validate operation completed successfully.
140219 11:22:45 mysqlbackup: INFO: Backup Image validation successful.
mysqlbackup: INFO: Source Image Path = /logs/img.bi
mysqlbackup completed OK!
```

  This is a sample output for a checksum mismatch in the header:

```
mysqlbackup: ERROR: Checksum mismatch.
Computed checksum: ###          Checksum in image: ### mysqlbackup:    ERROR: Problem verifying checksum of
Image Path = /meb/backups/image.mbi
mysqlbackup: ERROR: Backup image validation failed.
```

  This is a sample output for an image containing corrupted `.ibd` files:

```
mysqlbackup: ERROR: datadir/db2/bigtab1.ibd has corrupt page number : 64   page number from page header : 64
mysqlbackup: ERROR: datadir/db2/bigtab1.ibd is corrupt and has : 10 corrupt pages
mysqlbackup: ERROR: datadir/db2/t1.ibd has corrupt page number : 4   page number from     page header : 0
.......
```

```
mysqlbackup: ERROR: datadir/db2/t1.ibd is corrupt and has : 5 corrupt pages
mysqlbackup: ERROR: datadir/ibdata1 has corrupt page number : 63   page number from page header : 63
mysqlbackup: ERROR: datadir/ibdata1 has corrupt page number : 7   page number from page header : 7
..........
mysqlbackup: ERROR: datadir/ibdata1 is corrupt and has : 10 corrupt pages
mysqlbackup failed with errors!
```

This is a sample output for a successful validation for a compressed backup directory

```
mysqlbackup: INFO: /backups/backup-dir/datadir/tpch/tabnorm5.ibz Validated...
mysqlbackup: INFO: /backups/backup-dir/datadir/tpch/tabnorm6.ibz Validated...
mysqlbackup: INFO: /backups/backup-dir/datadir/tpch/tabnorm7.ibz Validated...
mysqlbackup: INFO: /backups/backup-dir/datadir/tpch/tabnorm8.ibz Validated...
mysqlbackup: INFO: /backups/backup-dir/datadir/tpch/tabnorm9.ibz Validated...
mysqlbackup: INFO: /backups/backup-dir/datadir/tpch/tabrowformat.ibz Validated...
140219 11:22:45 mysqlbackup: INFO: Validate backup directory operation completed successfully.
```

13.5 Single-File Backup Operations

To simplify transfer and management of backup data, you can keep each backup in a single file (the backup image). The `backup-to-image` command performs a backup directly to a single file, and there are the commands for packing an existing backup into a single file or unpacking a single-file backup to a full backup directory structure. These and other commands for working with single-file backups are explained below. For usage examples, see Section 4.3.5, “Making a Single-File Backup”.

```
mysqlbackup [STD-OPTIONS]
            [CONNECTION-OPTIONS]
            [SERVER-REPOSITORY-OPTIONS]
            [BACKUP-REPOSITORY-OPTIONS]
            [METADATA-OPTIONS]
            [COMPRESSION-OPTIONS]
            [SPECIAL-BACKUP-TYPES-OPTIONS]
            [INCREMENTAL-BACKUP-OPTIONS]
            [PARTIAL-BACKUP-RESTORE-OPTIONS]
            [SINGLE-FILE-BACKUP-OPTIONS]
            [PERFORMANCE-SCALABILITY-CAPACITY-OPTIONS]
            [MESSAGE-LOGGING-OPTIONS]
            [PROGRESS-REPORT-OPTIONS]
            [ENCRYPTION-OPTIONS]
            [CLOUD-STORAGE-OPTIONS]
            backup-to-image


mysqlbackup [STD-OPTIONS]
            [--backup-image=IMAGE] [--backup-dir=PATH]
            [MESSAGE-LOGGING-OPTIONS]
            [PROGRESS-REPORT-OPTIONS]
            [ENCRYPTION-OPTIONS]
            [CLOUD-STORAGE-OPTIONS]
            image-to-backup-dir

mysqlbackup [STD-OPTIONS]
            [--backup-dir=PATH] [--backup-image=IMAGE]
            [MESSAGE-LOGGING-OPTIONS]
            [PROGRESS-REPORT-OPTIONS]
            [ENCRYPTION-OPTIONS]
            [CLOUD-STORAGE-OPTIONS]
            backup-dir-to-image

mysqlbackup [STD-OPTIONS]
            [--backup-image=IMAGE]
            [MESSAGE-LOGGING-OPTIONS]
            [ENCRYPTION-OPTIONS]
            [CLOUD-STORAGE-OPTIONS]
            list-image
```

```
mysqlbackup [STD-OPTIONS]
            [--backup-image=IMAGE]
            [--backup-dir=PATH]
            [--src-entry=PATH] [--dst-entry=PATH]
            [MESSAGE-LOGGING-OPTIONS]
            [PROGRESS-REPORT-OPTIONS]
            [ENCRYPTION-OPTIONS]
            [CLOUD-STORAGE-OPTIONS]
            extract

mysqlbackup [STD-OPTIONS]
            [SERVER-REPOSITORY-OPTIONS]
            [--backup-image=IMAGE]
            [--backup-dir=PATH]
            [MESSAGE-LOGGING-OPTIONS]
            [PARTIAL-BACKUP-RESTORE-OPTIONS]
            [PROGRESS-REPORT-OPTIONS]
            [ENCRYPTION-OPTIONS]
            [CLOUD-STORAGE-OPTIONS]
            copy-back-and-apply-log
```

- `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 for the unpacking. For usage examples, see Section 4.3.5, “Making a Single-File Backup”.

  Note

  `image-to-backup-dir` only creates a raw backup directory, which is NOT ready to be restored by the `copy-back` command. To become a prepared backup, the backup directory has to go through an apply-log operation, executed either by a stand-alone `apply-log` command or as a part of a `copy-back-and-apply-log` command.

- `backup-dir-to-image`

  Packs an existing backup directory into a single file. The value for the `--backup-image` parameter should either be “`-`”(stands for standard output) or an absolute path outside of the `backup-dir` directory. 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. For usage examples, see Section 4.3.5, “Making a Single-File Backup”.

- `list-image`

  Display the contents of a single-file backup. Lists all files and directories in the image. For usage examples, see Section 4.3.5, “Making a Single-File Backup”.

  Note

  The `list-image` operation can be performed on a cloud backup only if the cloud proxy supports HTTP range headers.

- `extract`

  Unpacks individual files or directories from a single-file backup. It is useful for troubleshooting, or for restorations that do not require the full set of backup data. The resulting file or directory goes into the current directory, or into the backup directory, if specified with `--backup-dir`; in either case, the

destination directory must be empty. For usage examples, see Section 4.3.5, “Making a Single-File Backup”.

The `--src-entry=string` option can be used for selective extraction of files or directories whose path names in the image contain the *`string`* specified with the option.

Notes

- Some items are always extracted from the backup; see the descriptions of `--src-entry` for details.
- The option is currently not supported for the extraction of cloud backups, which can only be extracted in full.

The `--dst-entry=path` option, along with `--src-entry=path` option, can be used to extract files or directories into user-specified locations; see the description for the option for details.

The default destination for the extract is the current working directory. 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 pathnames on the local system even if the `--backup-dir` option is specified. The `--dst-entry` option must be used to relocate an absolute pathname; see Example 4.14, “Dealing with Absolute Path Names”.

Important

Even with all files extracted from the backup image, `extract` only creates a raw backup directory, which is NOT ready to be restored by the `copy-back` command. To become a prepared backup, the backup directory has to go through an apply-log operation, executed either by a stand-alone `apply-log` command or as a part of a `copy-back-and-apply-log` command.

- `copy-back-and-apply-log`

  See description for `copy-back-and-apply-log` in Section 13.3, “Restore Operations”.

Chapter 14 `mysqlbackup` Command-Line Options

Table of Contents

14.1 Standard Options 112
14.2 Connection Options 114
14.3 Server Repository Options 116
14.4 Backup Repository Options 118
14.5 Metadata Options 122
14.6 Compression Options 123
14.7 Incremental Backup Options 124
14.8 Partial Backup and Restore Options 127
14.9 Single-File Backup Options 133
14.10 Performance / Scalability / Capacity Options 136
14.11 Message Logging Options 143
14.12 Progress Report Options 144
14.13 Encryption Options 148
14.14 Cloud Storage Options 148
14.15 Options for Special Backup Types 151

The following sections describe the command-line options for the different modes of operation of `mysqlbackup`.

The table below list all the command options for `mysqlbackup`. Use the hyperlinks at the option names to jump to the detailed descriptions for the options.

Note

The command options can also be specified in configuration files; see explanations in Chapter 15, *Configuration Files and Parameters*. `mysqlbackup` follows the 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.

Table 14.1 List of All Options

Option Name	Description	Introduced
--backup-dir	The directory to store the backup data.	
--backup-image	Specifies the path name of the backup image.	
--backup_innodb_checksum_algorithm	The name of the checksum algorithm used for validating InnoDB tablespaces.	
--backup_innodb_data_file_path	Specifies Innodb system tablespace files' path and size in backup.	
--backup_innodb_data_home_dir	Backup base directory for all InnoDB data files in the system tablespace.	

Option Name	Description	Introduced
--backup_innodb_log_file_size	The size in bytes of each InnoDB backup log file.	
--backup_innodb_log_files_in_group	Number of InnoDB log files in backup.	
--backup_innodb_log_group_home_dir	Backup directory for InnoDB log files.	
--backup_innodb_page_size	The page size for all InnoDB tablespaces in a MySQL instance.	
--backup_innodb_undo_directory	The relative or absolute directory path where InnoDB creates separate tablespaces for the undo logs.	
--backup_innodb_undo_logs	Number of rollback segments in the system tablespace that InnoDB uses within a transaction.	
--backup_innodb_undo_tablespaces	The number of tablespace files that the undo logs are divided between when a non-zero innodb_undo_logs setting is used.	
--character-sets-dir	Directory for character set files.	
--cloud-access-key-id	AWS access key ID for logging onto Amazon S3.	
--cloud-aws-region	Region for Amazon Web Services that mysqlbackup access for S3.	
--cloud-bucket	The storage bucket for the backup image.	
--cloud-ca-info	Absolute path to the CA bundle file for host authentication for SSL connections.	3.12.3
--cloud-ca-path	CA certificate directory, in addition to the system's default folder.	3.12.3
--cloud-container	The Swift container for the backup image.	
--cloud-identity-url	The URL of the Keystone identity service.	
--cloud-object	The storage object for the backup image.	
--cloud-object-key	The Amazon S3 object key for the backup image.	
--cloud-password	Password for user specified by --cloud-user-id.	
--cloud-proxy	Proxy address and port number for overriding the environment's default proxy settings for accessing cloud service.	

Option Name	Description	Introduced
--cloud-region	The Keystone region for the user specified by --cloud-user-id.	
--cloud-secret-access-key	AWS secret access key.	
--cloud-service	Cloud service for data backup or restoration.	
--cloud-tempauth-url	The URL of the identity service for authenticating user credentials with Swift's TempAuth authentication system.	
--cloud-tenant	The Keystone tenant for the user specified by --cloud-user-id.	
--cloud-trace	Print trace information for cloud operations.	
--cloud-user-id	User ID for accessing Swift.	
--comments	Specifies comments string.	
--comments-file	Specifies path to comments file.	
--compress	Create backup in compressed format.	
--compress-level	Specifies the level of compression.	
--compress-method	Specifies the compression algorithm.	
--connect-if-online	Use connection only if available.	
--connect_timeout	Connection timeout in seconds.	
--databases	[Legacy] Specifies the list of non-InnoDB tables to back up.	
--databases-list-file	[Legacy] Specifies the pathname of a file that lists the non-InnoDB tables to be backed up.	
--datadir	Path to mysql server data directory.	
--debug	Print debug information.	
--decrypt	Decrypt backup image written in an MEB Secure File.	
--default-character-set	Set the default character set.	
--defaults-extra-file	Read this file after the global files are read.	
--defaults-file	Only read default options from the given file.	
--defaults-group-suffix	Also read option groups with the usual names and a suffix of str.	
--disable-manifest	Disable generation of manifest files for a backup operation.	

Option Name	Description	Introduced
--dst-entry	Used with single-file backups to extract a single file or directory to a user-specified path.	
--encrypt	Encrypt backup image and write it in an MEB Secure File.	
--exclude-tables	Exclude in a backup or restore tables whose names match the regular expression REGEXP.	
--exec-when-locked	Execute the specified utility in the lock phase near the end of the backup operation.	
--force	Force overwriting of data, log, or image files, depending on the operation.	
--free-os-buffers	Free filesystem cache by syncing the buffers	3.12.3
--help	Display help.	
--host	Host name to connect.	
--include	[Legacy] Backup only those per-table innodb data files which match the regular expression REGEXP.	
--include-tables	Include in a backup or a restore tables whose names match the regular expression REGEXP.	
--incremental	Specifies that the associated backup or backup-to-image operation is incremental.	
--incremental-backup-dir	Specifies the location for an incremental directory backup.	
--incremental-base	The specification of base backup for --incremental option.	
--incremental-with-redo-log-only	Specifies the incremental backup of InnoDB tables to be based on copying redo log to the backup, without including any InnoDB data files in the backup.	
--innodb_checksum_algorithm	The name of the checksum algorithm used for validating InnoDB tablespaces.	
--innodb_data_file_path	Specifies InnoDB system tablespace files' path and size.	
--innodb_data_home_dir	Specifies base directory for all InnoDB data files in the shared system tablespace.	

Option Name	Description	Introduced
--innodb_log_file_size	The size in bytes of each InnoDB log file in the log group.	
--innodb_log_files_in_group	The number of InnoDB log files.	
--innodb_log_group_home_dir	The directory path to InnoDB log files.	
--innodb_page_size	The page size for all InnoDB tablespaces in a MySQL instance.	
--innodb_undo_directory	The directory path to InnoDB undo tablespaces.	
--key	The symmetric key used for encryption and decryption.	
--key-file	The pathname of a file that contains the symmetric key used for encryption and decryption.	
--limit-memory	The memory in MB available for the MEB operation.	
--lock-wait-timeout	Specify the timeout in seconds for the FLUSH TABLES WITH READ LOCK statement that mysqlbackup issues during the final stage of a backup.	3.12.4
--log-bin-index	Specifies the absolute path of the index file that lists all the binary log files.	
--login-path	Read options from the named login path in the .mylogin.cnf login file.	
--master-info-file	Specifies the absolute path of the information file in which a replica records information about its source (for offline backups of replica servers only).	
--messages-logdir	Specifies the path name of an existing directory for storing the message log.	
--no-connection	Do not connect to server.	
--no-defaults	Do not read default options from any given file.	
--no-history-logging	Disable history logging even if connection is available.	
--no-locking	Disable all locking of tables during backups.	
--number-of-buffers	Specifies the exact number of memory buffers to be used for the backup operation.	

Option Name	Description	Introduced
--on-disk-full	Specifies the behavior when a backup process encounters a disk-full condition.	
--only-innodb	Back up only InnoDB data and log files.	
--only-innodb-with-frm	[Legacy] Back up only InnoDB data, log files, and the .frm files associated with the InnoDB tables.	
--only-known-file-types	Includes only files of a list of known types in the backup.	
--optimistic-busy-tables	Perform an optimistic backup, using the regular expression specified with the option to select tables that will be skipped in the first phase of an optimistic backup.	
--optimistic-time	Perform an optimistic backup with the value specified with the option as the optimistic time—a time after which tables that have not been modified are believed to be inactive tables.	
--page-reread-count	Maximum number of page re-reads.	
--page-reread-time	Wait time before a page re-read.	
--password	Connection password.	
--pipe	alias for –protocol=pipe.	
--port	TCP portnumber to connect to.	
--print-defaults	Print a list of option values supplied by defaults files and exit.	
--process-threads	Specifies the number of process-threads for the backup operation.	
--progress-interval	Interval between progress reports in seconds.	
--protocol	Connection protocol.	
--read-threads	Specifies the number of read-threads for the backup operation.	
--relay-log-index	Specifies the absolute path of the index file that lists all the relay log files.	
--relaylog-info-file	Specifies the absolute path of the information file in which a replica records information about the relay logs (for offline backups of replica servers only).	

Option Name	Description	Introduced
--rename	Rename a single table when it is selected by the --include-tables option to be restored	
--safe-slave-backup-timeout	When backing up a replica server, the timeout value for waiting for the replication SQL thread to drop its temporary tables.	3.12.3
--sbt-database-name	Used as a hint to the Media Management Software (MMS) for the selection of media and policies for tape backup.	
--sbt-environment	Comma separated list of environment variable assignments to be given to the SBT library.	
--sbt-lib-path	Path name of the SBT library used by software that manages tape backups.	
--secure-auth	Refuse client connecting to server if it uses old (pre-4.1.1) protocol.	
--shared-memory-base-name	It designates the shared-memory name used by a Windows server to permit clients to connect using shared memory (Windows only).	
--show-progress	Instructs mysqlbackup to periodically output short progress reports known as progress indicators on its operation.	
--skip-binlog	Do not include binary log files during backup, or do not restore binary log files during restore.	
--skip-final-rescan	Skip the final rescan for InnoDB tables that are modified by DDL operations.	3.12.4
--skip-messages-logdir	Disable logging to teelog file.	
--skip-relaylog	Do not include relay log files during backup, or do not restore relay log files during a restore.	
--skip-unused-pages	Skip unused pages in tablespaces when backing up InnoDB tables.	
--slave-info	Capture information needed to set up an identical replica server.	
--sleep	Time to sleep in milliseconds after copying each 1MB of data.	
--socket	Socket file to use to connect.	
--src-entry	Identifies a file or directory to extract from a single-file backup.	

Option Name	Description	Introduced
--ssl	Enable SSL for connection (automatically enabled with other --ssl- flags).	
--ssl-ca	CA file in PEM format (implies –ssl).	
--ssl-capath	CA directory (check OpenSSL docs,implies --ssl).	
--ssl-cert	X509 cert in PEM format (implies --ssl).	
--ssl-cipher	SSL cipher to use (implies --ssl).	
--ssl-key	X509 key in PEM format (implies --ssl).	
--ssl-verify-server-cert	Verify server's "Common Name" in its cert against hostname used when connecting.	
--start-lsn	Specifies the highest LSN value included in a previous backup.	
--suspend-at-end	Pauses the mysqlbackup command when the backup procedure is close to ending.	
--trace	Trace level of messages by mysqlbackup.	
--uncompress	Uncompress a backup during an operation.	
--use-tts	Enable selective backup of InnoDB tables using transportable tablespaces (TTS).	
--user	Database user name to connect.	
--verbose	Print more verbose information.	
--version	Display version information.	
--with-timestamp	Create a subdirectory underneath the backup directory with a name formed from the timestamp of the backup operation.	
--write-threads	Specifies the number of write-threads for the backup operation.	

14.1 Standard Options

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

Here is a list of the standard options:

- 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, System Variable,

and Status 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 optio
--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, System Variable, and Status 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.
  ```

- More standard options are available for `mysqlbackup`:

  `--verbose`: Print more verbose information.

  `--debug`=*`STRING`*: Print additional debug information. The option accepts the following arguments:

  - `all`: Print additional debug information for all operations
  - `sbt`: Print additional debug information for operations using the System Backup to Tape (SBT) interface
  - null: When a null string or no argument at all is specified for the option, `mysqlbackup` behaves as if the `--verbose` option is used.

  `--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 following:

Warning

For any restore operations, do NOT attempt to restore data to a non-empty data directory using the `--force` option; doing so may cause data corruption and other unexpected behaviors. Do not use the `--force` option with a `copy-back`

or a `copy-back-and-apply-log` operation, except for the special cases described below.

- Overwriting of InnoDB data and log files during the `apply-log` and `apply-incremental-backup` operations.
- When restoring a TTS backup, changing temporarily the value of `innodb_file_format` on the server, in order to allow restores of per-table InnoDB data files regardless of their format.
- Replacing of an image file during an `backup-to-image` or `backup-dir-to-image` operation.
- Restoring a partial image backup created with MySQL Enterprise Backup 3.11 or earlier; the `--force` option is required, due to a known issue (Bug# 20485910).
- Restoring a backup onto a server where the directory pointed to by the `.bl` file in the backup (a copy of the `.isl` file from the backed-up server) already contains `.ibd` data files.

`--trace=`*`level`*

Command-Line Format	`--trace=LEVEL`
Type	Enumeration
Default Value	`0`

Valid Values	`0`
	`1`
	`2`
	`3`

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

- 0 - INFO (information, warnings, errors)
- 1 - FINE (more information given than at trace level 0)
- 2 - FINER (finer level of information given than at trace level 1)
- 3 - FINEST (finest level of information that can be given)

14.2 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 Using Command Options 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. `mysqlbackup` always connects to the local server's IP address.

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

--login-path=name
--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 [ Enable 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 `mysqlbackup` 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.

14.3 Server Repository Options

These repository options specify various parameters related to the database server, from which the data is backed up or to which a backup is restored.

These options are used only with the following operations:

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

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.

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

- `datadir`=*`PATH`*

  This is the `datadir` value used by the MySQL instance. The `.frm` files reside 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`=*`PATH`*

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

  This parameter, together with `innodb_data_file_path`=*`SIZE`*, 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, the 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`=*`VALUE`*

  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) reside 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 file name begins with a `/` character or not, the files are located relative to the `innodb_data_home_dir` value.

- `innodb_log_group_home_dir`=*`PATH`*

  Specifies where InnoDB logs reside 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, the path is taken to be 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`=*`N`*

  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`=*`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.

- `innodb_page_size`=*`SIZE`*

  Specifies the page size for all InnoDB tablespaces.

  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_checksum_algorithm`=*`NAME`*

  Specifies the name of the checksum algorithm used for validating InnoDB tablespaces. Default is `innodb`.

  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_undo_directory`=*`PATH`*

  Specifies where the InnoDB undo log reside within the server repository. Usually the same as `datadir`, but can be different.

  For backups: Typically, you do not need to specify this option, because its value is retrieved automatically using the database connection. Specifies the option for an offline backup if the InnoDB undo log files do not reside under the server's data directory.

  For restores: The directory where InnoDB undo log files are to be restored. Specify the option only if the undo log files are to be restored outside of the server's data directory.

  Its value is derived as follows:

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

Warning

When using this option, make sure the undo log location does not change between successive restores of a full and an incremental backups, or of two incremental backups. Otherwise, the restore is going to fail.

14.4 Backup Repository Options

These options specify various parameters related to the backup directory and its layout, or to how the backup will be restored. Typically, `--backup-dir` is the only option from the group that you need to specify when using `mysqlbackup`.

The backup repository options are used with the following operations:

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

- Restore operations: `copy-back`, `copy-back-and-apply-log`.

The backup repository options are divided into two groups: the first one determines the structure of the backup, and the second one provides information on the original structure of the data on the backed-up server for future operations on the backup.

The following options determine the structure of the backup:

- `--backup_dir=`*`PATH`*

  Same as `--backup-dir`. The directory under which the backup data and metadata are stored, permanently or temporarily. It is a crucial parameter required for most kinds of backup and restore operations.

  The option is used differently for different operations and under different situations:

  - *For backup to a directory:* Use `--backup-dir` to specify the directory to store the backup data and metadata (including the `mysqlbackup` message log, the start and end LSN, and so on). The directory specified by `--backup-dir` cannot be a subdirectory of the directory specified by `--datadir`.

    When the `--with-timestamp` option is also specified, an additional level of subdirectory, with the timestamp in its name, is created under `--backup-dir` (see description for the `--with-timestamp` option for details). Unless the `--with-timestamp` option is used, the directory specified by `--backup-dir` must be empty, or the backup operation will fail with an error.

  - *For backup to a single file (including incremental, compressed, encrypted, and cloud backups):* Use `--backup-dir` to supply a temporary folder to save the backup metadata (including the `mysqlbackup` message log, the start and end LSN, and so on) and some temporary output. The backup data, together with a copy of the metadata, will be stored in a singe file whose name is specified with the `--backup-image` option. Note that, however, if `--backup-image` does not give a full path name, `mysqlbackup` will actually take the value of `--backup-image` as a path relative to the directory specified by `--backup-dir`, and thus store the single-file backup under `--backup-dir` (or, if the `--with-timestamp` option is used, under a subdirectory created under `--backup-dir`, which bears the timestamp in its name).

  - *For restoring a backup directory:* Use `--backup-dir` to specify the location of the backup directory, from which data will be restored to the server.

  - *For restoring a single-file backup (including incremental, compressed, encrypted, and cloud backups):* When using `copy-back-and-apply-log` to restore a single-file backup, use `--backup-dir` to supply a temporary folder to store the temporary data of the restore operation. The directory specified by `--backup-dir` should be empty—if a non-empty directory is used, the restore operation will still be carried out, but the restore data might be corrupted.

    When restoring a single-file backup created with the option setting `use-tts`=`with-minimum-locking`, the folder specified with `--backup-dir` is also used for extracting temporarily all the tables in the backup and for performing an `apply-log` operation to make the data up-to-date before restoring them to the server's data directory.

- `backup_innodb_data_home_dir=`*`PATH`*

  The directory under which the backup's InnoDB data files are to be stored. Specify the option if you want to put the data files at somewhere other than the default location (which is *`backup-dir`*`/datadir`). If the value of the parameter is different from *`backup-dir`*`/datadir`, it is stored into the `backup-my.cnf` file as `innodb_data_home_dir` for information, so that `mysqlbackup` can understand the structure of the backup when it performs various operations on the backup. Together

with `backup_innodb_data_file_path` option, it determines the actual file paths of the InnoDB data files inside the backup.

The value for the parameter is derived as follows:

- If `backup_innodb_data_home_dir` is not specified, its value will be *`backup-dir`*`/datadir`.
- If `backup_innodb_data_home_dir` is an absolute path, its value is used as-is.
- If `backup_innodb_data_home_dir` is a relative path, the path is taken to be relative to (that is, underneath) *`backup-dir`*.
- An empty string ("") for `backup_innodb_data_home_dir` means the value of `innodb_data_file_path` is to be taken as an absolute path..

This parameter is applicable only for backup operations; during a restore, the InnoDB data files are restored under the data directory specified by `--datadir`, unless another location is specified using the `--innodb_data_home_dir` option during restore.

- `backup_innodb_data_file_path`=*`VALUE`*

  The 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 are stored within the backup repository. Any file path specified with this option is taken to be relative to the value of the `backup_innodb_data_home_dir` option (that is true even if the file path is specified in the form of an absolute path, like `/abs/path/ibdata1:32M:autoextend`). To specify truly absolute paths for InnoDB data files in the backup with this option, you must set the `backup_innodb_data_home` option to `""` [empty string], in addition to using an absolute path for this option.

  When the parameter is not specified, it inherits the value from the value of the `innodb_data_file_path` option on the backed-up server. If both the source and destination of the backup attempt to use the same absolute paths that resolves to the same files, the backup is cancelled.

  The value of the parameter is stored into the `backup-my.cnf` file as `innodb_data_file_path` for information, so that `mysqlbackup` can understand the structure of the backup when it performs various operations on the backup.

- `backup_innodb_log_group_home_dir`=*`PATH`*

  The directory under which the backup's InnoDB logs will be stored. Specify this option only if you want to put the logs at somewhere other than the default location (which is *`backup-dir`*`/datadir`). If the value of the parameter is different from *`backup-dir`*`/datadir`, it is stored in the `backup-my.cnf` file as `innodb_log_group_home_dir` for information, so that `mysqlbackup` can understand the structure of the backup when it performs various operations on the backup. Note that while you can specify a directory for saving the logs, the names of the log files are fixed and not reconfigurable.

  This parameter is applicable only for backup operations; during a restore, the InnoDB log files are restored under the data directory specified by `--datadir`, unless another location is specified using

the `--innodb_log_group_home_dir` option during restore. The value of the parameter is derived as follows:

- If `backup_innodb_log_group_home_dir` is not specified, its value will be *`backup-dir/datadir`*.
- If `backup_innodb_log_group_home_dir` is an absolute path, its value is used as-is.
- If `backup_innodb_log_group_home_dir` is a relative path, the path is taken to be relative to (that is, underneath) *`backup-dir`*.
- An empty string ("") for the option produces an error.

- `backup_innodb_undo_directory`=*`PATH`*

  The relative or absolute directory path where separate tablespaces are created for the InnoDB undo logs during the backup. When unspecified, the option takes up the same value as `backup_innodb_log_group_home_dir`; specify this option only if you want to put the undo logs at some other location. If the value of the parameter is different from *`backup-dir`*`/datadir`, it is stored in the `backup-my.cnf` file as `innodb_undo_directory` for information, so that `mysqlbackup` can understand the structure of the backup when it performs various operations on the backup.

  This parameter is applicable only for backup operations; during a restore, the InnoDB undo log tablespaces are restored under the data directory specified by `--datadir`, unless another location is specified by the `--innodb_undo_directory` option during restore.

- `--with-timestamp`

  Creates a subdirectory underneath the backup directory, with a name formed with the timestamp of the backup operation. It is useful for maintaining a single backup directory containing many backup snapshots put under different subdirectories.

  Default: no timestamped subdirectory is created. To reuse the same backup directory for a new backup without using this option, either remove the previous backup files manually or, for a single-file backup, specify the `--force` option to overwrite the old backup file.

The following parameters provide information on the original structure of the data on the backed-up server for future operations on the backup, but do not affect the structure of the backup itself:

- `backup_innodb_log_files_in_group`=*`N`*

  The number of InnoDB log files in a log group on the restored server. See the description for `innodb_log_files_in_group` in the MySQL server manual. The value for this parameter, saved as `innodb_log_files_in_group` in the `backup-my.cnf` file, is derived as follows:

  - Use the `backup_innodb_log_files_in_group` value from command line or configuration file, if specified.
  - Else, use the `innodb_log_files_in_group` value from the backed-up server, if it is an online backup.
  - Else, use the `innodb_log_files_in_group` value from the `mysqlbackup` command line or configuration file.

- `backup_innodb_log_file_size`=*`SIZE`*

The maximum single InnoDB log file size in backup before switching to next log file, on the restored server. See the description for `innodb_log_file_size` in the MySQL server manual. The value for this parameter, saved as `innodb_log_file_size` in the `backup-my.cnf` file, is derived as follows:

- Use the `backup_innodb_log_file_size` value from command line or configuration file, if specified.
- Else, use the `innodb_log_file_size` value from the backed-up server, if it is an online backup.
- Else, use the specified `innodb_log_file_size` value from the `mysqlbackup` command line or configuration file.

- `backup_innodb_page_size`=*`SIZE`*

  Specifies, for an offline backup, the page size for all InnoDB tablespaces on the restored server. This option should be specified carefully, because the page size must be the same as that of the backed up MySQL instance, or the backup could become useless. For an online backup, the value is taken from the value of the `innodb_page_size` option on the backed-up server.

  Value of this option is saved in the `backup-my.cnf` file, to be used for restoring the database.

- `backup_innodb_undo_logs`=*`NUMBER`*

  Specifies, for an offline backup, the number of rollback segments in the InnoDB system tablespace on the restored server. This option should be specified carefully, because the value must be the same as that of `innodb_undo_logs` on the backed-up MySQL instance, or the backup could become useless. For an online backup, the value for the parameter is taken from the value of the `innodb_undo_logs` option on the backed-up server.

- `backup_innodb_undo_tablespaces`=*`NUMBER`*

  Specifies the number of tablespace files that the undo logs are divided between, when you use a non-zero `backup_innodb_undo_logs` setting. This option should be specified carefully, because the value must be the same as that of `innodb_undo_tablespaces` on the backed-up MySQL instance, or the backup could become useless. For an online backup, the value for the parameter is taken from the value of the `innodb_undo_tablespaces` option on the backed-up server. By default, all the undo logs are part of the system tablespace, and the system tablespace will always contain one undo tablespace in addition to those configured by innodb_undo_tablespaces.

- `backup_innodb_checksum_algorithm`=*`NAME`*

  Specifies, for an offline backup, the name of the checksum algorithm used for validating the InnoDB tablespaces on the restored server. This option should be specified carefully, because the checksum algorithm must be the same use on the backed-up MySQL instance, or the backup could become useless. For an online backup, the value is taken from the value of the `innodb_checksum_algorithm` option on the backed-up server.

  Default value of the option is “innodb”.

  Value of this option is saved in the `backup-my.cnf` file, to be used for restoring the database.

14.5 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 11.3, “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`*

Command-Line Format	`--comments=STRING`
Type	String

  Specifies 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`*

Command-Line Format	`--comments-file=PATH`
Type	File name

  Specifies 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.

14.6 Compression Options

For an overview on backup compression, see Section 4.3.3, “Making a Compressed Backup”.

- `--compress`

  Create backup in compressed format. For a regular backup, among all the storage engines supported by MySQL, only data files of the InnoDB format are compressed, and they bear the `.ibz` extension after the compression. Similarly, for a single-image backup, only data files of the InnoDB format inside the backup image are compressed. The binary log and relay log files are compressed and saved with the `.bz` extension when being included in a compressed backup.

  Default: compression is disabled.

- `--compress-method`=*`ALGORITHM`*

Command-Line Format	`--compress-method=ALGORITHM`
Type	Enumeration
Default Value	`lz4`

Valid Values	`zlib`
	`lz4`
	`lzma`
	`punch-hole`

	`none`

Specifies the compression algorithm. The supported arguments for the option and the algorithms they represent are:

- `lz4`: LZ4 r109. Out of the three algorithms that are supported, this is the most efficient one, typically taking the shortest backup and restore times with the lowest CPU cost. See lz4—Extremely Fast Compression algorithm for more details, including a comparison with other compression algorithms.
- `lzma`: LZMA 9.20. Out of the three supported algorithms, this typically provides the highest compression ratio; but it is also far more expensive in terms of CPU cost than the other two options. Thus we do not recommend this for active systems, but only for off-hour or inactive databases, or where I/O rates are extremely low.
- `zlib`: ZLIB v1.2.3. This is in between the other two supported algorithms in terms of both speed and compression ratio. ZLIB was the only compression algorithm available for MySQL Enterprise Backup versions prior to 3.10.

Default: lz4. Explicitly specifying a value for the option through a configuration file or command line automatically enables the `--compress` option.

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

Command-Line Format	`--compress-level=LEVEL`
Type	Numeric
Default Value	`1`
Minimum Value	`0`
Maximum Value	`9`

Specifies the level of compression, ranging from “0” to “9”: “0 ”disables compression; “1” is fastest compression, and “9” is highest (and slowest) compression. The option is only meaningful for compression using the ZLIB or LZMA algorithm; it is ignored when any other algorithms are selected by the `--compress-method` option.

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

- `--uncompress`

When used with the `apply-log` or `copy-back-and-apply-log` operation, uncompresses a compressed backup before applying the InnoDB log. When used with the `copy-back` operation, uncompresses a compressed prepared backup (created by the `backup-and-apply-log` command with the `--compress` option) before restoring it to a server (only supported for MySQL Enterprise Backup 3.12.3 and later).

14.7 Incremental Backup Options

For an overview of incremental backups and usage examples for these options, see Section 4.3.2, “Making a Differential or Incremental Backup”.

To take an incremental backup, specify the `--incremental` or `--incremental-with-redo-log-only`, along with the `--backup-dir` option. All InnoDB data modified after the specified LSN is copied in

the incremental backup. Depending on whether `--incremental` or `--incremental-with-redo-log-only` is used, 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 `--start-lsn` option.

  Only InnoDB tables are backed up incrementally. By default, all non-InnoDB and `.frm` files are included into the incremental backup and in their fullness. To exclude non-InnoDB data in an incremental backup, use the `--only-innodb` option.

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

  Specifies that an incremental backup is to be created using only the redo log. This alternate type of incremental backup has different performance characteristics and operational limitations comparing to backups created with the `--incremental` option; see Creating Incremental Backups Using Only the Redo Log for a discussion on their differences.

  To use this option, you also need to specify the `--incremental-base` option or the `--start-lsn` option. Just like with the `--incremental` option, only InnoDB tables are backed up incrementally. By default, all non-InnoDB and `.frm` files are included in incremental backup and in their fullness. To exclude non-InnoDB data in an incremental backup, use the `--only-innodb` option.

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

Command-Line Format	`--incremental-base=mode:argument`
Type	String

  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 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 non-TTS backup as recorded in the `backup_history` table of the server instance that is being backed up.

  Note

  If the last full or partial backup made was a TTS backup, `mysqlbackup` skips it, and keeps searching the backup history until it finds the last non-TTS backup and then returns its `end_lsn` value.

  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 for a subsequent incremental backup using this mode of the `--incremental-base` option.

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

Command-Line Format	`--start-lsn=LSN`
Type	Numeric

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.

Note

No binary log files are copied into the incremental backup if the `--start-lsn` option is used. To include binary log files for the period covered by the incremental backup, instead of `--start-lsn`, use the `--incremental-base` option, which provides the necessary information for `mysqlbackup` to ensure that no gap exists between binary log data included in the previous backup and the current incremental backup.

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

Specifies the location for data of an incremental directory backup. When creating or restoring an incremental directory backup, the option serves the same function as `--backup-dir` does for backups and restores in general, and the option can in fact be used interchangeably with `--backup-dir` for directory backups. See the description for `--backup-dir` for details.

For an `apply-incremental-backup` operation, the option specifies the incremental backup directory whose data is used to update a directory backup specified by the `--backup-dir` option.

Note

Do not use this option with any operations for image backups, for which the option has no meaning.

14.8 Partial Backup and Restore Options

Note

Since MySQL Enterprise Backup 3.10, the two options `--include-tables` and `--exclude-tables` have been introduced. These were intended for replacing the older options of `--include`, `--databases`, `--databases-list-file`, and `--only-innodb-with-frm`, which are incompatible with the new options and will be deprecated in future releases. For references purpose, we have included information on the older options at the end of this section in Legacy Partial Backup Options.

To select specific data to be backed up or restored, use the partial backup and restore options described in this section.

For an overview of partial backup and usage examples on the following options, see Section 4.3.4, “Making a Partial Backup”. See also Section 5.2.4, “Restoring Backups Created with the `--use-tts` Option”, on selective restore of tables from a backup.

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

Command-Line Format	`--include-tables=REGEXP`
Type	String

Include for backup or restoration only those tables (both Innodb and non-Innodb) whose fully qualified names (in the form of *`db_name.table_name`*) match the regular expression *`REGEXP`*. The regular expression syntax used is the extended form specified in the POSIX 1003.2 standard. For example, `--include-tables=^mydb\.t[12]$` matches the tables `t1` and `t2` in the database `mydb`. On Unix-like systems, quote the regular expression appropriately to prevent interpretation of shell meta-characters. `mysqlbackup` throws an error when the option is used without a regular expression being supplied with it.

While `mysqlbackup` understands the MySQL convention of quoting the database or the table name (or both) by backticks (see Schema Object Names), there is no need to include the backticks in the regular expression for `--include-tables`.

While the option can be used for different kinds of backups, selective restore is only supported for backups created using transportable tablespace (TTS) (that is, backups created with the `--use-tts` option). The option can also be used with the `backup-dir-to-image` and `image-to-backup-dir` commands to select tables when creating or unpacking a backup image.

The option cannot be used together with the legacy `--include`, `--databases`, `--databases-list-file`, or `--only-innodb-with-frm` option.

When used together with the `--exclude-tables` option, `--include-tables` is applied first, meaning `mysqlbackup` first selects all tables specified by `--include-tables` and then excludes from the set those tables specified by `--exclude-tables`.

- `--exclude-tables`=*`REGEXP`*

Command-Line Format	`--exclude-tables=REGEXP`
Type	String

Exclude for backup or restoration all tables (both Innodb and non-Innodb) whose fully qualified names (in the form of *`db_name.table_name`*) match the regular expression *`REGEXP`*. The regular

expression syntax is the extended form specified in the POSIX 1003.2 standard. For example, `--exclude-tables=^mydb\.t[12]$` matches the tables `t1` and `t2` in the database `mydb`. On Unix-like systems, quote the regular expression appropriately to prevent interpretation of shell meta-characters. `mysqlbackup` throws an error when the option is used without a regular expression being supplied with it.

While `mysqlbackup` understands the MySQL convention of quoting the database or the table name (or both) by backticks (see Schema Object Names), there is no need to include the backticks in the regular expression for `--exclude-tables`.

While the option can be used for different kinds of backups, selective restore is only supported for backups created using transportable tablespaces (TTS) (that is, backups created with the `--use-tts` option). The option can also be used with the `backup-dir-to-image` and `image-to-backup-dir` commands to select tables when creating or unpacking a backup image.

The option cannot be used together with the `--include`, `--databases`, `--databases-list-file`, or `--only-innodb-with-frm` option.

When used together with the `--include-tables` option, `--include-tables` is applied first, meaning `mysqlbackup` first select all tables specified by `--include-tables`, and then exclude from the set those tables specified by `--exclude-tables`.

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

  For back up only. By default, all files in the database subdirectories under the data directory of the server 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, `mysqlbackup` only backs up those types of files that are data files for MySQL or its built-in storage engines, which, besides the `ibdata*` files, have the following extensions:

  - `.ARM`: ARCHIVE table metadata
  - `.ARZ`: ARCHIVE table data
  - `.CSM`: CSV table metadata
  - `.CSV`: CSV table data
  - `.frm`: table definitions
  - `.ibd`: InnoDB tablespace created using the file-per-table mode
  - `.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`

  For back up only. When this option is used, only InnoDB data and log files are included in the backup, and 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.

  The option is not compatible with the `--slave-info` option.

- `--use-tts`[={`with-minimum-locking|with-full-locking`}]

Command-Line Format	`--use-tts[={with-minimum-locking|with-full-locking}]`
Type	Enumeration
Default Value	`with-minimum-locking`

Valid Values	`with-minimum-locking`
	
	`with-full-locking`

  Enable selective backup of InnoDB tables using transportable tablespaces (TTS). This is to be used in conjunction with the `--include-tables` and `--exclude-tables` options for selecting the InnoDB tables to be backed up by regular expressions. Using TTS for backups offers the following advantages:

  - Backups can be restored to a different server
  - The system tablespace is not backed up, saving disk space and I/O resources
  - Data consistency of the tables is managed by MySQL Enterprise Backup

  However, the option has the following limitations:

  - Supports only MySQL version 5.6 and after (as earlier versions of MySQL do not support TTS)
  - Can only backup tables that are stored in their own individual tablespaces (i.e., tables created with the innodb_file_per_table option enabled)
  - Non-InnoDB tables are not backed up
  - Cannot back up partitioned tables
  - Cannot be used for incremental backups
  - Does not include the binary log or the relay log in the backup

  See also Appendix B, *Limitations of MySQL Enterprise Backup* for some more minor limitations.

  There are two possible values for the option:

  - `with-minimum-locking`: Hot copies of the selected tables are backed up, and the tables are then locked in read-only mode while the redo log (with only the portion containing the relevant changes made after the hot backup) is being included in the backup. Any tables created during the locking phase are ignored.

- `with-full-locking`: The selected tables are locked in read-only mode while they are being backed up. The redo log is not included in the backup. Any tables created during the locking phase are ignored.

  Note

  Due to a known issue, when creating a backup using TTS for a server containing tables with a mix of the Antelope and Barracuda file formats, do NOT apply full locking on the tables.

  Default: `with-minimum-locking`

  To use the `--use-tts` option, extra privileges are required of the user through which `mysqlbackup` connects to the server; see Section 4.1.2, “Grant MySQL Privileges to Backup Administrator” for details.

  There are some special requirements for restoring backups created with the `--use-tts` option; see Section 5.2.4, “Restoring Backups Created with the `--use-tts` Option” for details.

- `--rename`=“*`old_table_name`* to *`new_table_name`*”

  Rename a single table when it is selected by the `--include-tables` or `--exclude-tables` option (or both together) to be restored to a database from a backup created using the `--use-tts` option. The table named *`old_table_name`* is renamed to *`new_table_name`*. Note that when using the option:

  - The `--include-tables` or `--exclude-tables` option (or both together) must be used in the restore command for the `--rename` option to work, unless there is only one table in the backup. Also, the `--include-tables` or `--exclude-tables` option (or both together) should specify one and only one table for restore when `--rename` is used, or the restore will fail.
  - *`old_table_name`* and *`new_table_name`* can be fully qualified (containing the database names, in the format of *`db_name`*.*`tb_name`*) or not. Regular expressions are not accepted for *`old_table_name`* and *`new_table_name`*.
  - The restore fails if *`old_table_name`* does not match with the table specified using the `--include-tables` or `--exclude-tables` option (or both together), or if *`new_table_name`* already exists in the target database.
  - The requirements listed in Section 5.2.4, “Restoring Backups Created with the `--use-tts` Option” apply.

  See Section 5.2.4, “Restoring Backups Created with the `--use-tts` Option”, for more information on selective restores, and for an example of table renaming.

Legacy Partial Backup Options

Important

Information in this subsection is only for using the legacy options of `--include`, `--databases`, `--databases-list-file`, and `--only-innodb-with-frm`, which will be deprecated in the upcoming issues. For creating partial backups, it is strongly recommended that the new options of `--include-tables` and `--exclude-tables` be used instead. Note that you cannot combine the legacy and the new partial-backup options in a single command.

Besides the legacy options, some other options are also discussed below, but the information is only for using the options together with the legacy partial-backup options.

For an overview of partial backups and usage examples for these legacy options, see Making a Partial Backup with the Legacy Options.

- `--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 used 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`. `mysqlbackup` throws an error when the option is used without a regular expression being supplied with it.

  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.

  When no InnoDB table names match the specified regular expression, an error is thrown with a message indicating there are no matches.

  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 Making a Partial Backup with the Legacy Options 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 Making a Partial Backup with the Legacy Options 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 white spaces surrounding the database or table names, as the white spaces 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-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 `mysqlbackup` 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 Making a Partial Backup with the Legacy Options for instructions and examples.

The option is not compatible with the `--slave-info` option.

Default: backups include files from all storage engines.

- • `--use-tts`[={`with-minimum-locking|with-full-locking`}]

  Enable selective backup of InnoDB tables using transportable tablespaces (TTS). This is to be used in conjunction with the `--include` option, which selects the InnoDB tables to be backed up by a regular expression. Using TTS for backups offers the following advantages:

  - Backups can be restored to a different server
  - The system tablespace is not backed up, saving disk space and I/O resources
  - Data consistency of the tables is managed by MySQL Enterprise Backup

  See important discussions here on the limitations with using the `--use-tts` option.

  There are two possible values for the option:

  - `with-minimum-locking`: Hot copies of the selected tables are backed up, and the tables are then locked in read-only mode while the redo log (with only the portion containing the relevant changes made after the hot backup) is being included in the backup. Any tables created during the locking phase are ignored.
  - `with-full-locking`: The selected tables are locked in read-only mode while they are being backed up. The redo log is not included in the backup. Any tables created during the locking phase are ignored.

  Default: back up with minimum locking

  There are some special requirements for restoring backups created with the `--use-tts` option; see the explanations in Section 5.2, “Performing a Restore Operation” for details.

14.9 Single-File Backup Options

These options are associated with single-file backups. You use them in combination with the `mysqlbackup` commands `backup-to-image`, `image-to-backup-dir`, `backup-dir-to-image`, `list-image`, and `extract`. For usage examples, see Section 4.3.5, “Making a Single-File Backup”.

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

Command-Line Format	`--backup-image=IMAGE`
Type	File name

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

You can optionally prefix the image name with `file:` to signify a file I/O (the default). For tape backups, prefix the image name with `sbt:`. See Section 4.3.5.2, “Backing Up to Tape” for details about tape backups.

- `--src-entry=`*`STRING`*

Command-Line Format	`--src-entry=STRING`
Type	Path name

Identifies files or directories whose pathnames contain the STRING to be extracted from a single-file backup. This option is used with the `extract` command. Optionally, you can also specify the `--dst-entry` option to extract a file or directory to a location different from its original path name.

For example: `src-entry=d1/f2` extracts only one file, `f1`, while `src-entry=d1/` extracts the entire directory tree for the `d1` folder (notice the slash (`/`) at the end of the argument, without which all files or folders containing the string `d1` in their pathnames will be extracted).

Default: All entries are extracted.

Notes

- The following items are always extracted from the backup, irrespective of the value of `--src-entry` (and the locations of their extraction are unaffected by the `--dst-entry` option):
  - The file `backup-my.cnf`.
  - A `datadir` folder (which only contains items matched by the `--src-entry` option).
  - A `meta` folder, which contains the file `backup_variables.txt`, a log file for the extract operation, and also items matched by the `--src-entry` option.
- The option is currently not supported for the `extract` command for cloud backups, which can only be extracted in full.

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

Command-Line Format	`--dst-entry=PATH`
Type	Path name

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 entry selected from the backup image by `--src-entry`. 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 `datadir/pets/` directory in a single-file backup as `/pets-extracted/`, use a command like the following:

```
mysqlbackup --src-entry=datadir/pets/ \
  --dst-entry=/pets-extracted/ \
```

```
  --backup-image=/var/myimage.bki  extract
```

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

In case the argument for `--src-entry` matches multiple files or folders, they are all extracted into a folder whose pathname, relative to the destination folder, is given by the argument of `--dst-entry` (unless the argument specifies an absolute path).

Default: Original pathnames are used to create files under the destination folder.

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

Command-Line Format	`--sbt-database-name=NAME`
Type	String
Default Value	`MySQL`

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 4.3.5.2, “Backing Up to Tape” for usage details.

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

Command-Line Format	`--sbt-lib-path=PATH`
Type	File name

Path name of the SBT library used by the 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 4.3.5.2, “Backing Up to Tape” for usage details.

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

Command-Line Format	`--sbt-environment=VAR1=value1[,VAR2=value2[,...]]` `SBT API provider)`
Type	String

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 equal 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.

14.10 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`*

Command-Line Format	`--number-of-buffers=NUMBER`
Type	Numeric
Default Value	`14`
Minimum Value	`1`

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: currently 14.

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`*

Command-Line Format	`--read-threads=NUMBER`
Type	Numeric
Default Value	`1`
Minimum Value	`1`
Maximum Value	`15`

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

Default: currently 1. 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`*

Command-Line Format	`--process-threads=NUMBER`
Type	Numeric
Default Value	`6`
Minimum Value	`1`
Maximum Value	`15`

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

Default: currently 6. 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`*

Command-Line Format	`--write-threads=NUMBER`
Type	Numeric
Default Value	`1`
Minimum Value	`1`
Maximum Value	`15`

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

Specifies the number of threads to use for writing data to disk. This option applies to these kinds of operations: `copy-back`, `copy-back-and-apply-log`, `extract`, `backup-to-image`, `backup`, and `backup-and-apply-log`. Multiple write threads are supported for any write target that is seekable; `--write-threads` is forced to be 1 only when the write target is non-seekable (e.g., when the backup is

written to `stdout`, to tape, or to cloud storage). The option is ignored when used with other single-file backup operations like `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.

Default: 1.

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

Command-Line Format	`--limit-memory=MB`
Type	Numeric
Default Value	`100 for apply-log (without uncompression), 400 for other operations`
Minimum Value	`0`
Maximum Value	`999999`
Unit	megabyte

Specify maximum memory in megabytes that can be used by `mysqlbackup`. 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` not used with `--uncompress`, 400 for all operations (in megabytes).

The memory limit specified by this option also caps the number of 16MB buffers available for multithreaded processing. For example, with a 400 MB limit, the maximum number of buffers is 25 (except for a cloud backup, for which extra memory is needed, and the maximum number of buffers is 18). If additional buffers are required because you increased the values for `--read-threads`, `--process-threads`, `--write-threads`, and/or `--number-of-buffers`, increase the `--limit-memory` value proportionally.

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

Command-Line Format	`--sleep=MS`
Type	Numeric
Default Value	`0`
Unit	millisecond

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.

- `--lock-wait-timeout`

Command-Line Format	`--lock-wait-timeout=S`
Introduced	3.12.4
Type	Numeric
Default Value	`60`
Minimum Value	`1`
Unit	second

  Specify the timeout in seconds for the `FLUSH TABLES WITH READ LOCK` statement, which `mysqlbackup` issues during the final stage of a backup to temporarily put the database into a read-only state. If the timeout is exceeded, the statement is failed and the lock on the tables is released, so that queries held up by the lock can then be executed. `mysqlbackup` then retries the statement and continues with the backup. The timeout prevents the case in which a long query running on the server prevents the `FLUSH TABLES WITH READ LOCK` statement from finishing, holding up further queries and eventually bringing down the server. Default is 60. Minimum value is 1.

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

Command-Line Format	`--page-reread-time=MS`
Type	Numeric
Default Value	`100`
Unit	millisecond

  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`*

Command-Line Format	`--page-reread-count=number`
Type	Numeric
Default Value	`500`

  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}`

Command-Line Format	`--on-disk-full=option`
Type	Enumeration

Default Value	`abort`

Valid Values	`abort`
	`warn`
	`abort_and_remove`

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`.

- `--skip-unused-pages`

  Skip unused pages in tablespaces when backing up InnoDB tables. This option is applicable to the `backup` and `backup-to-image` operations, but not to incremental backups. The option is ignored by the `backup-and-apply-log` operation.

  Note that backups created with the `--skip-unused-pages` option cannot be restored using `copy-back-and-apply-log`.

  Unused pages are free pages often caused by bulk delete of data. By skipping the unused pages during backups, this option can reduce the backup sizes and thus the required disk space and I/O resources for the operations. However, subsequent `apply-log` operations on the backups will take more time to complete, as the unused pages are inserted back into the tables during the operations.

- `--skip-binlog`

  Skip including the binary log files in the backup during a backup operation, or skip copying the binary log files onto a server during a restore operation.

  Binary log files, together with the binary log index file, are included by default for all kinds of online backups (full, incremental, compressed, partial, single-file, etc.). See Table 1.1, “Files in a MySQL Enterprise Backup Output Directory”, for details. Use this option to skip backing up binary logs for the following situations:

  - If resource or performance issues arise.
  - If any binary log files are missing on the server you are backing up, in order to avoid `mysqlbackup` throwing an error for the missing files.
  - If you are making an incremental backup that is based on a backup (full or incremental) created using the `--no-locking` option, as binary log information will then be unavailable to `mysqlbackup` in that situation.

  The binary log files and the binary log index file, when included in a backup, are always copied into the restored server's data directory during a restore operation; if that is not the behavior you desire, use this option to skip the restoring of the binary log.

- `--skip-relaylog`

  When working with a replica server, skip including the relay log files in the backup during a backup operation, or skip copying the relay log files onto a server during a restore operation.

  Relay log files, together with the relay log index file and the `master.info` and the `slave.info` files, are included by default for all kinds of online backups (full, incremental, compressed, partial, single-file, etc.) of a replica server. See Section 1.4, “Files that Are Backed Up”, for details. Use this option to skip backing up relay logs if resource, performance, or other issues arise.

  Note

  If a user runs a FLUSH LOGS statement while backup is in progress for a replica, the backup process will fail. Use the `--skip-relaylog` option if you expect a FLUSH LOGS statement will be run during the backup and it is not necessary to include the relay logs in the backup.

  The relay log files and the files backed up together with them, when included in a backup, are always copied into the restored server's data directory during a restore operation; if that is not the behavior you desire, use this option to skip the restoring of the relay log.

- `--skip-final-rescan`

  Skip the final rescan for InnoDB tables that are modified by DDL operations after the database has been read-locked near the end of a backup operation. This potentially shortens the duration for the lock and reduces the backup's impact on the server's normal operation, especially when many tables are being backed up.

  Warning

  This option can cause an incomplete or inconsistent backup if, during the backup operation, DDL operations are executed on any InnoDB tables whose file-per-table tablespaces are outside the MySQL data directory (i.e., any InnoDB tables created using the `DATA DIRECTORY` table option).

  The option is ignored for backups using the `--incremental-with-redo-log-only` option and for non-backup operations.

- `--log-bin-index`[=*PATH*]

Command-Line Format	`--log-bin-index=FILENAME`
Type	File name
Default Value	`data_dir/host_name-bin.index`

  For MySQL 5.5 and earlier, as well as all offline backups: specify the absolute path (including file name and extension) of the index file on the MySQL server that lists all the used binary log files, if it is different from the default path given below, in order to include the binary log files in the backup.

  Default: *`data_dir/host_name`*`-bin.index`.

- `--relay-log-index`[=*PATH*]

Command-Line Format	`--relay-log-index=FILENAME`
Type	File name

Default Value	`data_dir/host_name-relay-bin.index`

For offline backups of replica servers only: specify the absolute path (including file name and extension) of the index file on the MySQL server that lists all the used relay log files, if it is different from the default path given below, in order to include the relay log files in the backup.

Default: *`data_dir/host_name`*`-relay-bin.index`.

- `--master-info-file`[=*`PATH`*]

Command-Line Format	`--master-info-file=FILENAME`
Type	File name
Default Value	`data_dir/master.info`

For offline backups of replica servers only: specify the absolute path (including file name and extension) of the information file in which a replica records information about its source, if it is different from the default path given below, in order to include the information file in the backup.

Default: *`data_dir/`*`master.info`.

- `--relaylog-info-file`[=*`PATH`*]

Command-Line Format	`--relaylog-info-file=FILENAME`
Type	File name
Default Value	`data_dir/relay-log.info`

For offline backups of replica servers only: specify the absolute path (including file name and extension) of the information file in which a replica records information about the relay logs, if it is different from the default path given below, in order to include the information file in the backup.

Default: *`data_dir/`*`relay-log.info`.

- `--optimistic-time`[=*`DATE-TIME`*]

Command-Line Format	`--optimistic-time=DATE-TIME`
Type	String
Default Value	`now`

Perform an optimistic backup with the value specified with the option as the “optimistic time”—a time after which the tables that have not been modified are taken as “inactive tables.” The “inactive tables”are believed to be unlikely to change during the backup process. The inactive tables are backed up in the optimistic phase of the backup, and all other tables are backed up in the normal phase. See Section 4.3.6, “Making an Optimistic Backup” for details on the concept, use cases, and command samples for an optimistic backup.

Accepted formats for specifying the option include:

- `now`: This includes all tables into the optimistic phase of the backup process. It is the default value for the option when no value is specified.
- *`{Number}{Unit}`*: Indicates the optimistic time as a time at a certain duration into the past. *`{Unit}`* can be any one of `years`, `months`, `hours`, and `minutes`. Some examples for option strings in this format include: `5years`, `2days`,`13months`, `23hours`, and `35minutes`.

- A date-time format in any of the following forms: `YYMMDD`, `YYYYMMDD`, `YYMMDDHHMMSS`, `YYYYMMDDHHMMSS`, `YY-MM-DD`, `YYYY-MM-DD`, `YY-MM-DD HH.MM.SS`, or `YYYYMMDDTHHMMSS` (where `T` is the character `T`).

When both the `optimistic-time` and the `optimistic-busy-tables` options are used and they come into conflict on determining which tables are to be backed up in the optimistic phase, `optimistic-busy-tables` takes precedence over `optimistic-time`.

- `--optimistic-busy-tables=`*`REGEXP`*

Command-Line Format	`--optimistic-busy-tables=REGEXP`
Type	String

Perform an optimistic backup, using the regular expression specified with the option to select tables that will be skipped in the first phase of an optimistic backup, because they are likely to be modified during the backup process. Tables whose fully qualified names (in the form of *`database_name`*.*`table_name`*) are matched by the regular expression are taken as “busy tables”, which will be backed up in the second or the “normal” phase of the backup. Tables whose fully qualified names are NOT matched by the regular expression are taken as “inactive tables”, which will be backed up in the first or the “optimistic” phase of the backup. See Section 4.3.6, “Making an Optimistic Backup” for details on the concept, use cases, and command samples for an optimistic backup.

MySQL Enterprise Backup will throw an error if the option is used but no regular expression is supplied with it.

When both the `optimistic-time` and the `optimistic-busy-tables` options are used and they come into conflict on determining which tables are to be “optimistic”, `optimistic-busy-tables` takes precedence over `optimistic-time`.

- `--free-os-buffers`

(For release 3.12.3 and later) Free the system buffer cache at the end of a backup operation by syncing all data from the buffer cache to the hard disk. Using the option might increase the backup time significantly for systems with slow storage devices and databases with many tables.

Default: Automatic syncing disabled.

14.11 Message Logging Options

`mysqlbackup` writes important progress and error information to the `stderr` stream. The information is often very valuable for tracking down problems that occur during an operation. Starting from MySQL Enterprise Backup 3.9, the output to the `stderr` stream is also saved to a log file by default (for most `mysqlbackup` operations), so that the error information can be easily accessed in any debug process.

The message logging works like a `tee` process on a Unix-like system, in which the output of a program is split to be both displayed and saved to a file. The log file thus produced is named in the following format: `MEB_`*`timestamp_operation`*`.log`, where *`operation`* is the `mysqlbackup` operation that was run (e.g., `backup`, `apply-log`, etc.), and *`timestamp`* is the date and time at which the operation was run. Here are some examples of names for the log files:

```
MEB_2013-06-24.16-32-43_backup.log
MEB_2013-06-28.11-07-18_apply_log.log
MEB_2013-06-29.10-08-06_list_image.log
```

The following options control the message logging function:

- `--skip-messages-logdir`

  Skip message logging. Logging is turned on by default (except for the `list-image` and `validate` operations; see the description for the `--messages-logdir` option for details), and it is turned off by this option.

- `--messages-logdir=`*`path`*

Command-Line Format	`--messages-logdir=PATH`
Type	Directory name
Default Value	`backup_dir/meta`

  Specifies the path name of an existing directory for storing the message log. If the specified directory does not exist, message logging fails and returns an error message. When this option is omitted, the default directory of *`backup_dir`*`/meta` is used, where *`backup_dir`* is the directory specified with the `--backup-dir` option.

Note

Use this option to turn on message logging for the `list-image` and `validate` operations. Message logging is turned off by default for the two operations, because they do not modify any files and a message log is usually not required for debugging them. And because the default path name of `backup_dir/meta` is not meaningful for the two operations, this option is required for both turning on message logging and for supplying the path name of a directory in which to save the log file. However, if the `--skip-messages-logdir` option is also specified, it takes precedence and message logging is skipped.

The following are some examples showing how the message logging is controlled.

This creates a log file for the `backup` operation in the directory `/home/backup_dir/meta` due to the default settings:

```
mysqlbackup -uroot --port=3306 --backup-dir=/home/backup_dir backup
```

This skips message logging for the `backup` operation:

```
mysqlbackup -uroot --port=3306 --backup-dir=/home/backup_dir \
      --skip-messages-logdir backup
```

This creates a log file for the `apply-log` operation in an existing directory named `/home/teelog_dir`, rather than the default location:

```
mysqlbackup -uroot --port=3306 --backup-dir=/home/backup_dir \
      --messages-logdir=/home/teelog_dir apply-log
```

This creates a log file for the `list-image` operation in an existing directory named `/home/teelog_dir`:

```
mysqlbackup -uroot --port=3306 --backup-image=/backup/my.mbi \
      --messages-logdir=/home/teelog_dir list-image
```

14.12 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}]`

Command-Line Format	`--show-progress[=destinations]`
Type	Enumeration

Valid Values	`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`*

Command-Line Format	`--progress-interval=SECONDS`
Type	Numeric
Default Value	2
Minimum Value	1

Maximum Value	100000
Unit	second

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.

14.13 Encryption Options

These options are for creating encrypted single-file backups and for decrypting them. See Chapter 8, *Encryption for Backups* for more details and usage examples for the encryption and decryption functions of MySQL Enterprise Backup.

- `--encrypt`

  Encrypt the data when creating a backup image by a `backup-to-image` operation, or when packing a backup directory into a single file with the `backup-dir-to-image` command. It cannot be used with the `backup` or `backup-and-apply-log` command.

- `--decrypt`

  Decrypt an encrypted backup image when performing an `extract`, `image-to-backup-dir`, or `copy-back-and-apply-log` operation. It is also used for performing a `validate` or `list-image` operation on an encrypted backup image.

  The option cannot be used in a `apply-log`, `backup-and-apply-log`, or `copy-back` operation. For restoration using the `copy-back` command, the encrypted backup image has to be unpacked and decrypted first using the `image-to-backup-dir` or `extract` command, together with the `--decrypt` option.

- `--key`=*`STRING`*

Command-Line Format	--key=KEY
Type	String

The symmetric key for encryption and decryption of a backup image. It should be a 256-bit key, encoded as a string of 64 hexadecimal digits. See Chapter 8, *Encryption for Backups* on how to create a key. The option is incompatible with the `--key-file` option.

- `--key-file`=*`PATH`*

Command-Line Format	--key-file=FILE
Type	File name

The pathname to file that contains a 256-bit key, encoded as a string of 64 hexadecimal digits, for encryption and decryption of a backup image. The option is incompatible with the `--key` option.

14.14 Cloud Storage Options

These options are for using cloud storage for single-file operations. See Section 4.3.5.3, “Backing Up to Cloud Storage”, and Section 5.2.6, “Restoring a Backup from Cloud Storage to a MySQL Server”, for more information and instructions on using cloud storage with MySQL Enterprise Backup.

- `--cloud-service`=*`SERVICE`*

Cloud service for data backup or restoration. Currently, there are two types of cloud storage services supported by `mysqlbackup`, represented by the following values for the options:

- `openstack`: OpenStack Swift or compatible object storage services (for example, Oracle Cloud Infrastructure Object Storage and Oracle Cloud Infrastructure Object Storage Classic).
- `s3`: Amazon Simple Storage Service (S3).

Note

Due to some issues, Amazon S3 is currently not supported by MySQL Enterprise Backup 3.12.

- `--cloud-trace`

  Print trace information for cloud operations. It works independently of `--trace`, which specifies the trace level for the non-cloud operations of `mysqlbackup`. Any non-zero value for the option enables the trace function.

  Default value is “0.”

- `--cloud-proxy`=*`proxy-url:port`*

  Proxy address and port number for overriding the environment's default proxy settings for accessing a cloud storage service.

Note

The `list-image` operation can be performed on a cloud backup only if the cloud proxy supports HTTP range headers.

- `--cloud-ca-info`=*`PATH`*

  (For release 3.12.3 and later) Absolute path to the CA bundle file for host authentication for SSL connections. When the option is specified, the usage of the CA bundle file is preferred over the usage of individual `.pem` files for host authentication. .

- `--cloud-ca-path`=*`PATH`*

  (For release 3.12.3 and later) CA certificate directory, in addition to the system's default folder.

- Options used only for OpenStack Swift (using them when the argument for `--cloud-service` is anything other than `openstack` will cause `mysqlbackup` to throw an error):

  - `--cloud-container=`*`SWIFT_CONTAINER`*

    The Swift container for the backup image. For Oracle Cloud Infrastructure (OCI) Object Storage, this is the object storage bucket.

  - `--cloud-object=`*`SWIFT_OBJECT`*

    The Swift object for the backup image. Note that names of objects within the same container (or bucket, for OCI Object Storage) have to be unique.

  - `--cloud-user-id=`*`SWIFT_USER_ID`*

    User ID for accessing Swift. The user credentials are authenticated using the Swift TempAuth identity system when the `--cloud-tempauth-url` option is used and by the OpenStack Keystone identity service when the `--cloud-identity-url` option is used.

  - `--cloud-password=`*`SWIFT_PASSWORD`*

    Password for accessing Swift for the user specified by the `--cloud-user-id` option. The user credentials are authenticated using the Swift TempAuth identity system when the `--cloud-tempauth-url` option is used and by the OpenStack Keystone identity service when the `--cloud-identity-url` option is used.

  - `--cloud-tempauth-url=`*`SWIFT_TEMPAUTH-URL`*

    The TempAuth URL for authenticating user credentials. Either this option or `--cloud-identity-url` (but not both) should be used when accessing a Swift service.

  - `--cloud-identity-url=`*`SWIFT_KEYSTONE-URL`*

    The URL of the Keystone identity service, when it is used for authenticating user credentials. Either this option or `--cloud-tempauth-url` (but not both) should be used when accessing a Swift service.

  - `--cloud-tenant=`*`SWIFT_KEYSTONE-TENANT`*

    The Keystone tenant for the user specified by `--cloud-user-id`, when the Keystone identity service is used for authenticating user credentials.

  - `--cloud-region=`*`SWIFT_KEYSTONE-REGION`*

    The Keystone region for the user specified by `--cloud-user-id`, when the Keystone identity service is used for authenticating user credentials.

- Options used only for Amazon S3 (using them when the argument for `--cloud-service` is anything other than `s3` will cause mysqlbackup to throw an error):

  - `--cloud-bucket=`*`S3_BUCKET`*

    The storage bucket on Amazon S3 for the backup image.

    In order to perform cloud backups and restores with the bucket, the user identified by the `--cloud-access-key-id` option must have at least the following permissions on the bucket:

    - `s3:ListBucket`: For listing information on items in the bucket.
    - `s3:ListBucketMultipartUploads`: For listing multipart uploads in progress to the bucket.
    - `s3:GetObject`: For retrieving objects from the bucket.
    - `s3:PutObject`: For adding objects to the bucket.

  - `--cloud-object-key=`*`S3_OBJECT_KEY`*

    The Amazon S3 object key for the backup image.

  - `--cloud-access-key-id=`*`S3_KEY-ID`*

    AWS access key ID for logging onto Amazon S3.

  - `--cloud-secret-access-key=`*`S3_ACCESS-KEY`*

    AWS secret access key for the AWS access key id specified with `--cloud-access-key-id`.

  - `--cloud-aws-region=`*`S3_REGION`*

    Region for Amazon Web Services that `mysqlbackup` accesses for S3.

14.15 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 replica server, this option captures information needed to set up an identical replica 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 source server. This information is also printed in the `mysqlbackup` output. To set up a new replica for this source, restore the backup data on another server, start a replica 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 Replica” for instructions.

Note

Only use this option when backing up a replica server. Its behavior is undefined when used on a source or non-replication server.

This option is not compatible with the `--no-locking` option; using both options together will make `mysqlbackup` throw an error.

> This option is not compatible with the `--only-innodb` or `--only-innodb-with-frm` options.

- `--safe-slave-backup-timeout=SECONDS`

  (For release 3.12.3 and later) For a statement-based replication (SBR) or a mixed-based replication setup, the option specifies the time (in seconds) `mysqlbackup` will wait for `Slave_open_temp_tables` to become “0” (which is true when no temporary tables are open) to complete the backup for a replica server by asserting a global read lock and copies all the non-InnoDB tables. If the duration of the wait exceeds that specified with the option, `mysqlbackup` times out and throws an error. The wait is for preventing `mysqlbackup` from finishing a replica backup when there are temporary tables still open. See descriptions in Temporary tables on statement-based replication (SBR) replica for details on how `mysqlbackup` deals with temporary tables on a replica server.

  In addition, `mysqlbackup` also runs an initial check at the beginning of a replica backup to see if `Slave_open_temp_tables=0` becomes true within the duration set by `--safe-slave-backup-timeout`. If it does not, `mysqlbackup` takes it as an early sign that before the backup is completed, some temporary tables are likely to remain open after the timeout limit is exceeded; `mysqlbackup` then throws an error, instead of continuing with the backup. When that happens, you can either restart the backup with a higher value for `--safe-slave-backup-timeout`, or retry at a time when fewer temporary tables are being used.

  Default: 300

  Warning

  Proper setting of this value depends on the use case, and it can vary a lot according to the situation. Setting the value for this option either too high or too low will affect adversely the performance of the backup operation:

  - *Too high:* If you need to wait for a long time for there to be no more temporary tables, the chance is that the change rate for your database is quite high, which means the amount of redo log data to be included in the backup will be large and the restore time for the backup will be long. In such a case, it would have been better to have let `mysqlbackup` timeout and then restart the backup operation, so the tables are copied in their final states. It is therefore not helpful to set a high timeout value for the option. As a very general rule of thumb: even for busy databases that use many contemporary tables, do not set the value to more than an a few hours.
  - *Too low:* Setting the wait time value too low would make the backup process time out very easily and when that happens, the process has to be restarted. With a repeating cycle of restarts, the backup might then take a long time to complete, and resources used on the failed backups will be wasted. As a very general rule of thumb, do not set the timeout to below the default value of 300s.

  For a row-based replication (RBR) setup, temporary tables are not replicated onto the replica. Users who are certain that SBR is not occurring for the replica can set `--safe-slave-backup-timeout=0`, with which `mysqlbackup` will not check for any open temporary tables before finishing the backup.

- `--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` and `--suspend-at-end` to back up only certain InnoDB tables.

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

Command-Line Format	`--exec-when-locked="utility arg1 arg2 ..."`
Type	String

You can use this option to run a script that backs up any information that is not included as part of the usual backup. For example, with `--exec-when-locked`, you can use `mysqldump` to back up tables from the MEMORY storage engine, which are not on disk.

Set any variable you want to use within your script before you run `mysqlbackup`. In the following example, the `BACKUP_DIR` environment variable is set to point to the current backup directory (quotes are used for the argument of `--exec-when-locked`, to prevent premature expansion of the variable `BACKUP_DIR`):

On Unix or Linux systems:

```
export BACKUP_DIR=path_to_backupdir
mysqlbackup --exec-when-locked="mysqldump mydb t1 > $BACKUP_DIR/t1.sql" other_options mysqlbackup_comman
```

Or on Windows systems:

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

If the utility cannot be executed or returns a non-zero exit status, 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 the suspension is lifted.

Chapter 15 Configuration Files and Parameters

You can specify `mysqlbackup` options either on the command line or as configuration parameters inside a configuration file. This section describes the use of configuration files.

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.

Within `mysqlbackup` option names, dashes (`-`) and underscores (_) may be used interchangeably, similar to `mysqld` parameters that use this same convention (see Using Options on the Command Line in the MySQL Reference Manual for details). The MySQL server's reference manual typically lists the parameter names with underscores, to match the output of the `SHOW VARIABLES` statement.

Options Files

`mysqlbackup` 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 15.1 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.

Part IV Appendixes

Table of Contents

A Frequently Asked Questions for MySQL Enterprise Backup 161
B Limitations of MySQL Enterprise Backup 163
C Compatibility Information for MySQL Enterprise Backup 165
    C.1 Supported Platforms 165
    C.2 Cross-Platform Compatibility 165
    C.3 Compatibility with MySQL Versions 165
    C.4 Compatibility with Older Versions of MySQL Enterprise Backup 165
    C.5 Compatibility Notes for Specific MySQL Versions 165
D MySQL Enterprise Backup Release Notes 167
MySQL Enterprise Backup Glossary 169

Appendix A 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

- A.1: Does MySQL Enterprise Backup work with MySQL Server version *x.y.z*?
- A.2: What is the big `ibdata` file that is in all the backups?
- A.3: Can I back up non-InnoDB data with MySQL Enterprise Backup?
- A.4: What happens if the apply-log step is interrupted?
- A.5: Why is the option `--defaults-file` not recognized?
- A.6: Can I back up a database on one OS platform and restore it on another one using MySQL Enterprise Backup?
- A.7: What if I have included the binary log or relay log in my backup but do not want to restore it?

Questions and Answers

A.1: Does MySQL Enterprise Backup work with MySQL Server version *x.y.z*?

See Section C.3, "Compatibility with MySQL Versions" for details of compatibility between different releases of MySQL Enterprise Backup and MySQL Server.

A.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.

A.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.

A.4: What happens if the apply-log step is interrupted?

If `mysqlbackup` 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.

A.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.

A.6: Can I back up a database on one OS platform and restore it on another one using MySQL Enterprise Backup?

See Section C.2, “Cross-Platform Compatibility” for details.

A.7: What if I have included the binary log or relay log in my backup but do not want to restore it?

If you want to skip the restore of the binary log, relay log, or both during a restore, use the `--skip-binlog` option, the `--skip-relaylog` option, or both with your `copy-back` or `copy-back-and-apply-log` command.

Appendix B Limitations of MySQL Enterprise Backup

Please refer to the MySQL Enterprise Backup version history in Appendix D, *MySQL Enterprise Backup Release Notes* for a list of fixed `mysqlbackup` bugs. Here are a list of limitaions of MySQL Enterprise Backup:

- 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 C.5, “Compatibility Notes for Specific MySQL Versions” for details.
- 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 binary log position is updated. The binary log 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 by, for example, 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 Section 7.1, “Optimizing Backup Performance” for guidelines about backup timing and performance.
- Do not run the DDL operations (for example, `ALTER TABLE`, `TRUNCATE TABLE`, `OPTIMIZE TABLE`, `REPAIR TABLE`, `RESTORE TABLE` or `CREATE INDEX`) while a backup operation is going on. The resulting backup might become corrupted.
- The `engines` column in the `mysql.backup_history` table does not correctly reflect the storage engines of the backed-up databases.
- Hot backups for large databases with heavy writing workloads (say, in the order of gigabytes per minute) can take a very long time to complete due to the huge redo log files that are generated on the server while the backup is running. However, when it is a relatively small subset of tables in the database that are being modified frequently, the Optimistic Backup feature can be used to improve performance and reduce backup size, as well as backup and recovery times. See Section 4.3.6, “Making an Optimistic Backup” for details.
- Compressed InnoDB tables from MySQL server 5.6.10 and earlier cannot be restored with MySQL Enterprise Backup 3.9.0 or later, due to a known issue with the InnoDB storage engine (see Bug# 72851 on the MySQL Bug System).
- While it is possible to backup to or restore from a Network Attached Storage (NAS) device using MySQL Enterprise Backup, due to networking issues that might arise, the consistency of the backups and the performance of the backup or restore operations might be compromised.
- When creating a backup using transportable tablespace (TTS) for a server containing tables with a mix of the Antelope and Barracuda file formats, do not apply full locking on the tables (that is, do not specify `--use-tts=with-full-locking`). Instead, just specify `--use-tts` or `--use-tts=with-minimum-locking`, both of which will apply minimum locking to the tables (Bug #20583946).
- Tables created on the MySQL server with the `ANSI_QUOTES` SQL mode cannot be backed up using transportable tablespace (TTS).
- When a file of an unrecognized file type exists under a subdirectory in the server's data directory, it will be backed up by `mysqlbackup` unless the `--only-known-file-types` option is used. However, if

the name of the file does not have an extension, it will cause `mysqlbackup` to throw an error when it tries to restore the backup to a server.

- If a table on the server is dropped while `mysqlbackup` is backing up that table, `mysqlbackup` throws an error and exits.
- Offline backups using MySQL Enterprise Backup 3.12.2 sometimes fail, with occasional crashes of `mysqlbackup`.
- Cloud operations by MySQL Enterprise Backup are not supported on macOS and Windows platforms, and also on Linux platforms when generic Linux builds are used for both the server and MySQL Enterprise Backup (i.e., when both the server and MySQL Enterprise Backup have been installed using generic Linux tarballs).
- Using the `--src-entry` option with the `extract` command on cloud backups will cause the command to fail. Cloud backups can only be extracted in full.
- Due to some issues, cloud backup and restore using Amazon S3 is currently not supported by MySQL Enterprise Backup 3.12.
- A compressed directory backup fails when a general tablespace bears the same basename as the database's system tablespace (usually `ibdata1`) and exists in the same directory with it (usually the server's data directory). A compressed single-file backup created under the same situation will be corrupted, and cannot be restored. To avoid the problem, the server administrator should not put into the same directory the system tablespace and a general tablespace of the same basename; if that is unavoidable, do not perform a compressed backup for the database.
- When working with a replication set up whose source server also belongs to a separate Group Replication setup, over time, create backups consistently either from the source or the replica, but not from both. Otherwise, there will be conflicts between the `id` values generated by the source and the replica, causing backups to fail.

Appendix C Compatibility Information for MySQL Enterprise Backup

Table of Contents

C.1 Supported Platforms 165
C.2 Cross-Platform Compatibility 165
C.3 Compatibility with MySQL Versions 165
C.4 Compatibility with Older Versions of MySQL Enterprise Backup 165
C.5 Compatibility Notes for Specific MySQL Versions 165

This section describes information related to compatibility issues for MySQL Enterprise Backup releases.

C.1 Supported Platforms

See Supported Platforms: MySQL Database (MySQL platform support evolves over time; please refer to the page for the latest updates).

C.2 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.

C.3 Compatibility with MySQL Versions

MySQL Enterprise Backup 3.12 supports MySQL 5.5 and 5.6.

C.4 Compatibility with Older Versions of MySQL Enterprise Backup

MySQL Enterprise Backup 3.12 can be used to restore the following kinds of backups created by earlier versions of the product:

- Backups created for MySQL 5.5 by MySQL Enterprise Backup 3.5 to 3.11.
- Backups created for MySQL 5.6 by MySQL Enterprise Backup 3.8 to 3.11.

Note

For any restore that involves a server upgrade or downgrade, see the important discussion in Section 5.4, “Restoring a Backup with a Database Upgrade or Downgrade”.

C.5 Compatibility Notes for Specific MySQL Versions

This section lists any performance-related features and settings in specific 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.

See MySQL Enterprise Backup 3.9 Release Notes for details on the fixes and enhancements related to these MySQL 5.6 features.

Appendix D MySQL Enterprise Backup Release Notes

Release notes for MySQL Enterprise Backup are published separately. See MySQL Enterprise Backup 3.12 Release Notes.

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.

.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.

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

.bz file
: When `mysqlbackup` performs a compressed backup for a server that has binary logging enabled, it transforms each binary log file and relay log file (for a **replica** server in a **replication** setting) to a *`binary-or-relay-log-file-name`*`.bz` file. The `.bz` files are uncompressed at the time of restore.
See Also binary log, .bz file, compression, compression level, .ibz file, relay log.

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, mysqldump, physical backup, warm backup.

backup directory
: The directory under which the backup data and metadata are stored, permanently or temporarily. It is used in most kinds of backup and restore operations, including single-file backups and restores. See the description of the `--backup-dir` option on how the backup directory is used for different purposes and for different operations.

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 replica 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 source server when taking a backup in a replication context, you can specify the `--slave-info` option.

The binary log, if enabled on the server, is backed up by default.
See Also binlog, relay log, 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 websites, 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, 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 macOS 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.

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, partial backup.

differential backup
A backup that captures only the data changed since the last full backup. It has the potential to be smaller and faster than a **full backup**, but is usually bigger and takes longer to create than an **incremental backup**. See Section 4.3.2, “Making a Differential or 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, incremental 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 **replica** 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, replica, replication, 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 an `.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.

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, 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, but is, nevertheless, still saved as an `.ibz` file in the compressed backup.
See Also .bz file, compression, compression level, .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 4.3.2, "Making a Differential or 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, 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.

.MYD file
: A file that MySQL uses to store data for a MyISAM table.
  See Also .MYI file.

.MYI file
: A file that MySQL uses to store indexes for a MyISAM table.
  See Also .MYD file.

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
: See source.

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 macOS 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, 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, 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, physical backup, restore.

N

non-TTS backup
: A backup that is NOT created using transportable tablespace (TTS), that is, not with the `--use-tts` option.
See Also transportable tablespace, TTS backup.

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.

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, 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.

optimistic backup
: Optimistic backup is a feature for improving performance for backing up and restoring huge databases in which only a small number of tables are modified frequently. An optimistic backup consists of two phases: (1) the optimistic phase in which tables that are unlikely to be modified during the backup process (identified by the user with the `optimistic-time` option or, by exclusion, with the `optimistic-busy-tables` option) are backed up without any locks on the MySQL instance; (2) a normal phase, in which tables that are not backed up in the first phase are being backed up in a manner similar to how they are processed in an ordinary backup: the InnoDB files are copied first, and then other relevant files and copied or processed with various locks applied to the database. The redo logs, undo logs, and the system tablespace are also backed up in this phase. See Section 4.3.6, “Making an Optimistic Backup” for details.

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, 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.

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 14.10, “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**. Related `mysqlbackup` options are `--include-tables`, `--exclude-tables`, `--use-tts`, `--only-known-file-types`, and `--only-innodb`.
See Also backup, database, 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**. Related `mysqlbackup` options are `--include-tables`, `--exclude-tables`, and `--rename`.
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.

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.

relay log
: A record on a **replica** server for the events read from the **binary log** of the source server and written by the replication I/O thread. The relay log, like the **binary log**, consists of a set of numbered files containing events that describe database changes, and an index file that contains the names of all used relay log files. For more information on relay log, see The Relay Log. The relay log on a server is backed up by default.
See Also binary log, replication.

replica
: In a **replication** configuration, a database server that receives updates from a **source** server. Typically used to service user queries, to minimize the query load on the source server. With **MySQL Enterprise Backup**, you might take a backup on one server, and restore on a different system to create a new replica server with the data already in place. You might also back up data from a replica server rather than the source, to minimize any slowdown of the overall system.
See Also replication, source.

replication
: A common configuration for MySQL deployments, with data and DML operations from a **source** server synchronized with a set of **replica** servers. With MySQL **Enterprise Backup,** you might take a backup on one server, and restore on a different system to create a new replica server with the data already in place. You might also back up data from a replica server rather than the source, to minimize any slowdown of the overall system.
See Also replica, source.

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.

selective backup
: Another name for **partial backup**
  See Also partial backup, selective restore.

selective restore
: Another name for **partial restore**
  See Also partial restore, selective backup.

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
: See replica.

source
: In a **replication** configuration, a database server that sends updates to a set of **replica** servers. It typically dedicates most of its resources to write operations, leaving user queries to the replicas. With **MySQL Enterprise Backup**, typically you perform backups on the replica servers rather than the source, to minimize any slowdown of the overall system.
  See Also replica, 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.

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.

transportable tablespace
: A feature that allows a **tablespace** to be moved from one instance to another. Traditionally, this has not been possible for InnoDB tablespaces because all table data was part of the **system tablespace**. In MySQL 5.6 and higher, the `FLUSH TABLES ... FOR EXPORT` syntax prepares an InnoDB table for copying to another server; running `ALTER TABLE ... DISCARD TABLESPACE` and `ALTER TABLE ... IMPORT TABLESPACE` on the other server brings the copied data file into the other instance. A separate `.cfg` file, copied along with the **.ibd file**, is used to update the table metadata (for example the **space ID**) as the tablespace is imported. See Importing InnoDB Tables for usage information.

  Use the `--use-tts` option to create a backup with transportable tablespace. See also Section 5.2.4, “Restoring Backups Created with the `--use-tts` Option”.
  See Also partial backup.

TTS
: Short form for **transportable tablespace**.

See Also partial backup, transportable tablespace.

TTS backup
: A backup that is created using transportable tablespace (TTS), that is, with the `--use-tts` option.
See Also non-TTS backup, partial backup, transportable 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 websites, you might prefer a **hot backup**.
See Also backup, cold backup, hot backup.

Index

Symbols

--exclude-tables option, 44
--include-tables option, 44
.frm file, 43

A

Antelope, 93, 169
apply, 169
apply-incremental-backup option, 58, 97
--apply-log option, 97
.ARM file, 169
.ARZ file, 169

B

backup, 169
backup directory, 169
backup option, 96
backup repository, 170
backup-and-apply-log option, 96
--backup-dir option, 119
backup-dir-to-image option, 102
backup-image option, 133
backup-my.cnf, 170
backup-my.cnf file, 7
backup-to-image option, 96, 133
backups
  cold, 5
  compressed, 6, 42, 47, 58, 123
  controlling overhead, performance, and scalability, 136
  encrypted, 79
  full, 37
  hot, 5
  incremental, 6, 38, 124
  InnoDB tables only, 93
  logical, 6
  message logging, 143
  optimistic, 53
  parallel, 6
  partial, 44, 127
  physical, 6
  prepared, 7, 57
  preparing to restore, 57
  progress report, 144
  raw, 7, 57
  scheduled, 55
  single-file, 6, 48
  streaming, 6, 50
  to cloud, 51
  to tape, 51, 81
  troubleshooting, 85
  uncompressed, 6, 47
  verifying, 34
  warm, 5
backup_content.xml, 7
backup_content.xml file, 88
backup_create.xml, 7
backup_create.xml file, 88
backup_history table, 86
backup_progress table, 86
backup_variables.txt file, 7
Barracuda, 93, 170
benchmarking, 73
binary log, 63, 170
binlog, 170
.bz file, 169

C

changelog, 167
changes
  release notes, 167
cloud backups, 51
--cloud-access-key-id option, 151
--cloud-aws-region option, 151
--cloud-bucket option, 151
--cloud-ca-info option, 149
--cloud-ca-path option, 149
--cloud-identity-url, 150
--cloud-object, 150
--cloud-object-key option, 151
--cloud-password, 150
--cloud-proxy option, 149
--cloud-region, 150
--cloud-secret-access-key option, 151
--cloud-service option, 149
--cloud-tempauth-url, 150
--cloud-tenant, 150
--cloud-trace option, 149
--cloud-user-id, 150
cold backup, 5, 170
command-line tools, 6
--comments option, 123
--comments-file option, 123
comments.txt file, 7, 123
--compress option, 42, 123
--compress-level option, 42, 124
--compress-method option, 124
compressed backups, 6, 42, 47, 58, 123
compression, 171
compression level, 171
configuration file, 171
configuration options, 155
connection, 171
connection options, 114

copy-back option, 20, 57, 98
copy-back-and-apply-log option, 35, 98
corruption problems, 85
--counter-container, 150
crash recovery, 57, 171
cron jobs, 55
.CSM file, 7
.CSV file, 7

D

data dictionary, 171
database, 172
--databases option, 131
--databases-list-file option, 132
datadir directory, 7
--datadir option, 116
--data_home_dir option, 116
--decrypt option, 148
decryption, 79
differential backup, 172
--disable-manifest option, 136
disk storage for backup data, 6, 50
distributed file system, 56
downtime, 172
--dst-entry option, 134

E

--encrypt option, 148
encrypted backups, 79
encryption, 79
error codes, 85
exclude, 172
--exclude-tables option, 127
--exec-when-locked option, 153
extract, 172
extract option, 102, 133

F

FAQ, 161
file format, 172
files backed up, 7
frequently asked questions, 161
.frm file, 7, 172
full backup, 37, 173

G

GRANT statement, 31

H

history table, 173
hot backup, 5, 173

I

ibbackup_logfile file, 7
.ibd file, 7, 173
ibdata file, 7, 173
ibreset command, 85
.bz file, 7
.ibz file, 7, 173
ib_logfile file, 7
image, 174
image-to-backup-dir option, 102, 133, 133
image_files.xml file, 7, 88
include, 174
--include option, 44, 131
--include-tables option, 127
incremental backup, 6, 124, 174
--incremental option, 125
--incremental-backup-dir option, 126
--incremental-base option, 125
--incremental-with-redo-log-only option, 125
InnoDB, 174
InnoDB tables, 5, 7, 93, 93
  compressed backup feature, 42
  incremental backup feature, 38
installing MySQL Enterprise Backup, 21
instance, 174

K

--key option, 148
--key-file option, 148

L

--limit-memory option, 138
list-image option, 102, 133
--lock-wait-timeout option, 139
locking, 174
log, 7, 96, 174
--log-bin-index, 141
logical backup, 6, 174
logs
  of backup operations, 86
LSN, 38, 124, 175

M

manifest, 7, 88, 136, 175
--master-info-file, 142
media management software, 175
media management software (MMS) products, 81
MEMORY tables, 55
message logging, 143
meta directory, 7
MMS products, 81
monitoring backup jobs, 83
.MRG file, 175

my.cnf, 175
my.ini, 175
.MYD file, 7
.MYD file, 175
.MYI file, 7
.MYI file, 175
MyISAM, 175
MyISAM tables, 93
MySQL Enterprise Backup, 176
mysqlbackup, 93, 176
  and media management software (MMS) products, 81
  configuration options, 155
  examples, 37
  files produced, 7
  modes of operation, 95
  options, 105
  overview, 6
  required privileges, 31
  using, 29
mysqlbinlog command, 63
mysqldump, 55, 176

N

--no-history-logging option, 123
--no-locking option, 139
non-TTS backup, 176
--number-of-buffers option, 136

O

offline, 176
--on-disk-full option, 140
online, 176
--only-innodb option, 129
--rename option, 130
--only-innodb option, 132
--only-known-file-types option, 128
.opt file, 7, 176
optimistic backup, 53, 142, 143, 176
--optimistic-busy-tables, 143
--optimistic-time, 142
options, mysqlbackup, 105
  connection, 114
  for cloud storage, 148
  for compression, 123
  for controlling backup overhead, performance, and scalability, 136
  for controlling message logging, 143
  for controlling progress reporting, 144
  for encryption, 148
  for generating metadata, 122
  for incremental backups, 124
  for partial backups, 127
  for single-file backups, 133
  for special types of backups, 151
  in configuration files, 155
  layout of backup files, 118
  layout of database files, 116
  modes of operation, 95
  options in common with mysql, 112
  standard options, 112
Oracle Secure Backup, 177
OS user, 32
OSB, 177

P

--page-reread-count option, 139
--page-reread-time option, 139
.par file, 7, 177
parallel backup, 73, 76, 177
parallel backups, 6
partial backup, 44, 127, 177
partial restore, 177
performance
  of backups, 73
  of restores, 76
performance of backup operations, 6
permissions, 32
physical backup, 6, 177
point in time, 177
point-in-time recovery, 63
posix_fadvise() system call, 6
prepared backup, 7, 57, 177
privileges, 31
--process-threads option, 137
progress indicator, 144
progress table, 177
--progress-interval, 148

R

RAID, 73, 76
raw backup, 7, 57, 178
--read-threads option, 136
redo log, 178
regular expression, 178
relay log, 178
--relay-log-index, 142
--free-os-buffers, 143
--relaylog-info-file, 142
release notes, 167
replica, 67, 69, 178
replication, 67, 69, 70, 178
repository, 178
restore, 178
restoring a backup, 57
  at original location, 35
  backup created with the --use-tts option, 61

examples, 58
mysqlbackup options, 97
overview, 20
point-in-time recovery, 63
preparation, 57
restore external tablespaces at different locations, 62
row format, 179

S

--safe-slave-backup-timeout, 152
SBT, 179
--sbt-database-name option, 135
--sbt-environment option, 135
--sbt-lib-path option, 135
selective backup, 179
selective restore, 179
server, 179
server repository, 179
--show-progress, 145
single-file backup, 6, 48, 101, 133, 179
--skip-binlog, 140
--skip-final-rescan, 141
--skip-relaylog, 141
--skip-unused-pages, 140
--slave-info option, 151
--sleep option, 138
source, 70, 179
space for backup data, 6
--src-entry option, 134
--start-lsn option, 126
storage access network, 56
streaming, 50, 179
streaming backups, 6
suspend, 180
--suspend-at-end option, 152
system backup to tape, 180
system tablespace, 7, 180

T

table, 180
tablespace, 180
tape backups, 51, 81
transportable tablespace, 180
.TRG file, 7
.TRG file, 180
.TRN file, 7
troubleshooting for backups, 85
TTS, 180
TTS backup, 181

U

--uncompress option, 124
uncompressed backups, 6, 47
--use-tts option, 129

V

validate option, 100
validating a backup, 99
verifying a backup, 34

W

warm backup, 5, 181
what is new, 23
--with-timestamp option, 121
--write-threads option, 137