

MySQL Enterprise Backup User's Guide (Version 8.0.21)

Abstract

This is the user manual for MySQL Enterprise Backup, a commercially licensed backup utility for MySQL databases. It explains the different kinds of backup and restore that can be performed with MySQL Enterprise Backup, and describes the commands for performing them. Strategies for optimizing backup and restore operations are also discussed.

For notes detailing the changes in each release, see the [MySQL Enterprise Backup 8.0 Release Notes](#).

For legal information, including licensing information, see the [Preface and Legal Notices](#).

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

Document generated on: 2020-07-22 (revision: 8134)

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 The mysqlbackup Client	5
1.2 Overview of Backup Types	5
1.3 Files that Are Backed Up	7
1.3.1 Types of Files Contained in a Backup	7
1.3.2 Files Backed up for InnoDB Data	19
1.3.3 Files Backed up for Data Stored with MyISAM and Other Storage Engines	20
1.3.4 Files Generated by mysqlbackup	20
1.4 The Backup Process	20
2 Installing MySQL Enterprise Backup	23
3 What's New in MySQL Enterprise Backup 8.0?	25
II Using MySQL Enterprise Backup	31
4 Backing Up a Database Server	35
4.1 Before the First Backup	35
4.1.1 Collect Database Information	35
4.1.2 Grant MySQL Privileges to Backup Administrator	36
4.1.3 Designate a Location for the Backup Directory	39
4.2 The Typical Backup / Verify / Restore Cycle	40
4.2.1 Backing Up an Entire MySQL Instance	40
4.2.2 Verifying a Backup	42
4.2.3 Restoring a Database	43
4.3 Backup Scenarios and Examples	46
4.3.1 Making a Single-File Backup	46
4.3.2 Making a Full Backup	50
4.3.3 Making a Differential or Incremental Backup	51
4.3.4 Making a Compressed Backup	57
4.3.5 Making a Partial Backup	57
4.3.6 Making an Optimistic Backup	60
4.3.7 Making a Back Up of In-Memory Database Data	63
4.3.8 Making Scheduled Backups	63
4.4 Making Backups with a Distributed File System (DFS) or Storage Access Network (SAN)	64
5 Recovering or Restoring a Database	65
5.1 Performing a Restore Operation	65
5.1.1 Restoring a Compressed Backup	66
5.1.2 Restoring an Encrypted Backup Image	67
5.1.3 Restoring an Incremental Backup	67
5.1.4 Table-Level Recovery (TLR)	68
5.1.5 Restoring Backups Created with the <code>--use-tts</code> Option	69
5.1.6 Restoring a Backup from Cloud Storage to a MySQL Server	69
5.1.7 Restoring External InnoDB Tablespaces to Different Locations	70
5.1.8 Advanced: Preparing and Restoring a Directory Backup	71
5.2 Point-in-Time Recovery	71
5.3 Restoring a Backup with a Database Upgrade or Downgrade	73
6 Working with Encrypted InnoDB Tablespaces	75
7 Backing up Using Redo Log Archiving	79
8 Using MySQL Enterprise Backup with Replication	81
8.1 Setting Up a New replica	81
8.2 Backing up and Restoring a Replica Database	83
8.3 Restoring a Source Database	83
8.4 Working with Encrypted Binary and Relay Logs	85
9 Using MySQL Enterprise Backup with Group Replication	87
10 Encryption for Backups	89

11	Using MySQL Enterprise Backup with Media Management Software (MMS) Products	91
11.1	Backing Up to Tape with Oracle Secure Backup	91
12	Using MySQL Enterprise Backup with Docker	95
13	Performance Considerations for MySQL Enterprise Backup	97
13.1	Optimizing Backup Performance	97
13.2	Optimizing Restore Performance	100
14	Monitoring Backups with MySQL Enterprise Monitor	103
15	Using MySQL Enterprise Backup with MySQL Enterprise Firewall	105
16	Troubleshooting for MySQL Enterprise Backup	107
16.1	Exit codes of MySQL Enterprise Backup	107
16.2	Working Around Corruption Problems	108
16.3	Using the MySQL Enterprise Backup Logs	109
16.4	Using the MySQL Enterprise Backup Manifest	110
III	<code>mysqlbackup</code> Command Reference	113
17	<code>mysqlbackup</code>	117
18	<code>mysqlbackup</code> commands	119
18.1	Backup Operations	119
18.2	Update Operations	120
18.3	Restore Operations	121
18.4	Validation Operations	124
18.5	Other Single-File Backup Operations	124
18.6	Other Operations	126
19	<code>mysqlbackup</code> Command-Line Options	129
19.1	Standard Options	134
19.2	Connection Options	136
19.3	Server Repository Options	137
19.4	Backup Repository Options	139
19.5	Metadata Options	142
19.6	Compression Options	143
19.7	Incremental Backup Options	145
19.8	Partial Backup and Restore Options	148
19.9	Single-File Backup Options	152
19.10	Performance / Scalability / Capacity Options	155
19.11	Message Logging Options	161
19.12	Progress Report Options	163
19.13	Encryption Options	166
19.14	Options for Working with Encrypted InnoDB Tablespaces and Encrypted Binary/ Relay Logs	166
19.15	Cloud Storage Options	167
19.16	Options for Special Backup Types	170
19.17	Other Options	173
20	Configuration Files and Parameters	175
IV	Appendixes	177
A	Frequently Asked Questions for MySQL Enterprise Backup	181
B	Limitations of MySQL Enterprise Backup	183
C	Compatibility Information for MySQL Enterprise Backup	185
C.1	Cross-Platform Compatibility	185
C.2	Compatibility with MySQL Versions	185
C.3	Compatibility with Older MySQL Enterprise Backup	185
D	Backup History Table Update	187
E	SBT Backup History Table Update	189
F	Backup Progress Table Update	191
	MySQL Enterprise Backup Glossary	193
	Index	205

List of Tables

1.1 Types of Files in a Backup	7
4.1 Information Needed to Back Up a Database	35
16.1 MySQL Enterprise Backup Exit Codes and Messages	107
19.1 List of All Options	129

List of Examples

4.1 Single-File Backup to Absolute Path	46
4.2 Single-File Backup to Relative Path	46
4.3 Single-File Backup to Standard Output	46
4.4 Convert Existing Backup Directory to Single Image	46
4.5 Extract Existing Image to Backup Directory	46
4.6 List Single-File Backup Contents	47
4.7 Validate a Single-File Backup	47
4.8 Extract Single-File Backup into Current Directory	47
4.9 Extract Single-File Backup into a Backup Directory	47
4.10 Selective Extract of Single File	47
4.11 Selective Extract of Single Directory	47
4.12 Dealing with Absolute Path Names	47
4.13 Single-File Backup to a Remote Host	48
4.14 Single-file Backup to a Remote MySQL Server	48
4.15 Stream a Backup Directory to a Remote MySQL Server	48
4.16 Creating a Cloud Backup on Oracle Cloud Infrastructure Object Storage Classic	49
4.17 Creating a Cloud Backup on Oracle Cloud Infrastructure Object Storage	49
4.18 Creating a Cloud Incremental Backup on Oracle Cloud Infrastructure Object Storage	49
4.19 Creating a Cloud Backup on an OpenStack Object Storage	49
4.20 Extract an Existing Image from an Oracle Cloud Infrastructure Object Storage Classic Container to a Backup Directory	50
4.21 Extract an Existing Image from Amazon S3 Cloud Storage to a Backup Directory	50
4.22 Making an Uncompressed Partial Backup of InnoDB Tables	59
4.23 Making a Compressed Partial Backup	60
4.24 Optimistic Backup Using the Option <code>optimistic-time=YYMMDDHHMMSS</code>	61
4.25 Optimistic Backup Using the Option <code>optimistic-time=now</code>	61
4.26 Optimistic Backup Using the <code>optimistic-busy-tables</code> Option	61
4.27 Optimistic and Partial Backup Using both the <code>optimistic-busy-tables</code> and <code>optimistic-time</code> Options	61
5.1 Restoring a Database	65
5.2 Restoring a Compressed Backup	66
5.3 Restoring a Compressed Directory Backup	66
5.4 Restoring a Compressed and Prepared Directory Backup	66
5.5 Restoring an Encrypted Backup Image	67
5.6 Restoring an Incremental Backup Image	67
5.7 Restoring A Selected Table from an Image Backup	69
5.8 Restoring Selected Tables in a Schema from an Image Backup	69
5.9 Restoring and Renaming a Table from a TTS Backup	69
5.10 Restoring a Single-file Backup from an Oracle Cloud Infrastructure (OCI) Object Storage Classic Container to a MySQL Server	70
5.11 Restoring a Single-file Backup from an Oracle Cloud Infrastructure (OCI) Object Storage to a MySQL Server	70
5.12 Restoring a Cloud Incremental Backup from an Oracle Cloud Infrastructure (OCI) Object Storage Service to a MySQL Server	70
5.13 Restoring a Single-file Backup from an OpenStack Object Storage to a MySQL Server	70
5.14 Restoring a Single-file Backup from Amazon S3 to a MySQL Server	70
5.15 Restoring a Backup Directory using <code>copy-back-and-apply-log</code>	71
5.16 Applying the Log to a Backup	71
5.17 Applying the Log to a Compressed Backup	71
11.1 Sample <code>mysqlbackup</code> Commands Using MySQL Enterprise Backup with Oracle Secure Backup	93
18.1 Apply Log to Full Backup	121
20.1 Sample <code>backup-my.cnf</code> file	175

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 8.0 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, 2020, 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 and Java 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

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	The mysqlbackup Client	5
1.2	Overview of Backup Types	5
1.3	Files that Are Backed Up	7
1.3.1	Types of Files Contained in a Backup	7
1.3.2	Files Backed up for InnoDB Data	19
1.3.3	Files Backed up for Data Stored with MyISAM and Other Storage Engines	20
1.3.4	Files Generated by mysqlbackup	20
1.4	The Backup Process	20
2	Installing MySQL Enterprise Backup	23
3	What's New in MySQL Enterprise Backup 8.0?	25

Chapter 1 Introduction to MySQL Enterprise Backup

Table of Contents

1.1 The mysqlbackup Client	5
1.2 Overview of Backup Types	5
1.3 Files that Are Backed Up	7
1.3.1 Types of Files Contained in a Backup	7
1.3.2 Files Backed up for InnoDB Data	19
1.3.3 Files Backed up for Data Stored with MyISAM and Other Storage Engines	20
1.3.4 Files Generated by mysqlbackup	20
1.4 The Backup Process	20

MySQL Enterprise Backup 8.0.21 is a backup utility for MySQL 8.0.21. It is a multi-platform, high-performance tool, offering rich features like “hot” (online) backup, incremental and differential backup, selective backup and restore, support for direct cloud storage backup, backup encryption and compression, and many other valuable features.

While optimized for use with InnoDB tables, MySQL Enterprise Backup is capable of backing up and restoring all kinds of tables created by any kinds of storage engines supported by MySQL. The parallelism of its read and write processes (performed in independent, multiple threads) and its block-level parallelism (different threads can read, process, or write different chunks within a single file) allow backup and restore processes to be completed with great speed, and often with a significant performance gain when compared to a [logical backup](#) using tools like [mysqldump](#).

MySQL Enterprise Backup is a valuable tool for maintaining and safeguarding your MySQL data, and for quick and reliable recovery when accidents or disasters strike. It is part of the MySQL Enterprise Edition, available to subscribers under a commercial license.

Among other things, this manual explains:

- How to install MySQL Enterprise Backup.
- The different kinds of backups that can be performed with MySQL Enterprise Backup, how to perform them, and some tips on choosing the right kind of backups for your system.
- How to restore backups created by MySQL Enterprise Backup.
- How to use MySQL Enterprise Backup in special situations or for special purposes (for example, setting up replication, using Media Management Software (MMS) products, or using a Distributed File System (DFS)).
- The [mysqlbackup](#) client, its commands and command options.

1.1 The mysqlbackup Client

All MySQL Enterprise Backup functions are executed with the [mysqlbackup](#) client. It is used for performing different types of backup and restore operations, as well as other related tasks like backup compression, decompression, validation, and so on.

Use of the [mysqlbackup](#) client and the related commands are explained and illustrated throughout this manual. For detailed information on the [mysqlbackup](#) commands and command options, see [Part III, “mysqlbackup Command Reference”](#).

1.2 Overview of Backup Types

When it comes to formulating your backup strategy, performance and storage space are the key considerations. You want the backup to complete quickly, with as little CPU overhead on the database server as possible. 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. Under such considerations, various strategies for backing up your database often give you different advantages, for the different trade-offs you make when choosing a particular strategy. To choose the strategy that best fits your needs, you have to understand the nature of each kind of backups that MySQL Enterprise Backup can perform, for which this section is giving a brief overview.

Kinds of backups according to the level of service disruption

Depending on how the database operations would be disrupted during a backup, the backup is classified as “hot,” “warm,” or “cold”:

- *Very Low to Low Level of Disruption:* A [hot backup](#) is a backup performed while the database is running. This type of backups does not block normal database operations. It captures even changes that occur while the backup is happening. Comparing to the other backup types, it causes the least disruption to the database server, and it is a desirable backup option when you want to avoid taking your application, web site, or web service offline. However, before a hot backup can be restored, there needs to be an extra process of preparing the backup to make it consistent (i.e., correctly reflecting the state of the state of the database at the time the backup was completed). See [Section 5.1.8, “Advanced: Preparing and Restoring a Directory Backup”](#) for more explanations.

When connected to a running MySQL server, MySQL Enterprise Backup performs hot backup for InnoDB tables.

- *Medium to High Level of Disruption:* A [warm backup](#) is a backup performed with the database put under a read-only state. This type of backups blocks any write operations to the tables during the backup process, but still allow tables to be read.

When connected to a running MySQL server, MySQL Enterprise Backup backs up all MyISAM and other non-InnoDB tables using the [warm backup](#) technique after all InnoDB tables have already been backed up with the [hot backup](#) method. Therefore, to back up as much data as possible during the hot backup phase, you should designate InnoDB as the default storage engine for new tables (which is the default setting for MySQL servers), or convert existing tables to use the InnoDB storage engine.

- *High to Very High Level of Disruption:* A [cold backup](#) is a backup created while the database is stopped. It is very disruptive to a database service. *MySQL Enterprise Backup 8.0 does not support cold backups.*

Kinds of backups according to whether all data, or recent changes only are backed up

According to whether you want to include all data into your backup or only the recent changes, and according to recent changes since when, you can perform either a full backup, a differential backup, or an incremental backup. The three types of backups have different levels of requirements for CPU overhead and disk space, thus are suitable for different situations:

- A [full backup](#) includes the complete data from the database (except in cases where some tables are excluded with the [partial backup options](#)).
- A [differential backup](#) includes all changes to the data since the last full backup. It is faster than a full backup, saves storage space on the database server, and saves on network traffic when the backup is being transferred to a different server. However, it requires additional processing to make the backup ready for restore, which you can perform on a different system to minimize CPU overhead on the database server.
- An [Incremental backup](#) includes all changes to the data since the last backup. It offers similar advantages over a full backup as a differential backup does, and often to a even greater extent by further decreasing the backup size. But it might also require more preparations on a longer series of backups, before a restore can be performed.

Compressed versus uncompressed backups

Backup compression saves you storage space and network traffic to transfer the backup data onto a different server. Compression does add some CPU overhead, but the overhead is algorithm dependent and it is fairly low for the default algorithm used by MySQL Enterprise Backup. Also, compression often reduces the IO overhead by a great deal, which might shorten the restore time, especially for slower IO devices. However, during the restore process, you need time for decompression and also storage space for both the compressed and decompressed data at the same time. So, take into account the additional storage space and the extra time needed during a restore when considering whether to create compressed backups.

When streaming backup data to another server, you might want to compress the backup either 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.

For more on techniques and tradeoffs involving backup and restore performance, see [Chapter 13, Performance Considerations for MySQL Enterprise Backup](#).

1.3 Files that Are Backed Up

This section explains the various types of files contained in a backup.

1.3.1 Types of Files Contained in a Backup


The following table shows the different types of files that are included in a single-file backup image or a directory backup. In the case of a single-file backup, unpack the file into a [backup directory](#) structure using the [extract](#) or the [image-to-backup-dir](#) command to view the files.

Table 1.1 Types of Files in a Backup

File Name, Pattern, or Extension	Relation to Original Data Files	Notes
<code>ibdata*</code>	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.
<code>*.ibd</code>	An InnoDB tablespace, which can be (a) a file-per-table tablespace, containing a single InnoDB table and associated indexes, or (b) a file-per-table external tablespace located outside of the server's data directory, containing a single InnoDB table and associated indexes, or (c) a general tablespace , containing one or more tables and their 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.
<code>*.ibz</code>	Compressed form of InnoDB data files from the MySQL data directory.	Produced instead of <code>.ibd</code> files in a compressed backup. The <code>ibdata*</code> files representing the InnoDB system tablespace also receive this extension in a compressed backup. The <code>.ibz</code> files are uncompressed during the apply-log , copy-back , or

File Name, Pattern, or Extension	Relation to Original Data Files	Notes
		copy-back-and-apply-log step.
*.sdi	Hold Serialized Dictionary Information (SDI) for MyISAM tables, which is the tables' metadata.	The database is put into a read-only state while these files are copied. These files are copied unmodified.
*.MYD	MyISAM table data.	The database is put into a read-only state while these files are copied. These files are copied unmodified.
*.MYI	MyISAM index data.	The database is put into a read-only state while these files are copied. These files are copied unmodified.
*.CSM	Metadata for CSV tables.	These files are copied unmodified. 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 unmodified. 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 unmodified.
*.opt	Database configuration information.	The database is put into a read-only state while these files are copied. These files are copied unmodified.
*.ARM	Archive storage engine metadata.	The database is put into a read-only state while these files are copied. These files are copied unmodified.
*.ARZ	Archive storage engine data.	The database is put into a read-only state while these files are copied. These files are copied unmodified.
backup-my.cnf	Records the configuration parameters that specify the layout of the MySQL data files.	Used in restore operations to reproduce the same layout as when the backup was taken.


File Name, Pattern, or Extension	Relation to Original Data Files	Notes
<code>ibbackup_ibd_files</code>	Records names of the <code>.ibd</code> 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.
<code>ibbackup_logfile</code>	A condensed version of the <code>ib_logfile*</code> files from the MySQL data directory.	The InnoDB log files (<code>ib_logfile*</code>) 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 <code>ibbackup_logfile</code> , and used to re-create the <code>ib_logfile*</code> files during the apply-log phase.
<code>ibbackup_redo_log_only</code>	Created instead of the <code>ibbackup_logfile</code> for incremental backups taken with the <code>--incremental-with-redo-log-only</code> option.	
<code>ib_logfile*</code>	Created in the backup directory by <code>mysqlbackup</code> during the <code>apply-log</code> 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 <code>apply-log</code> phase after the initial backup, using the changes recorded in the <code>ibbackup_logfile</code> file.
Timestamped directory, such as <code>2011-05-26_13-42-02</code>	Created by the <code>--with-timestamp</code> option. All the backup files go inside this subdirectory.	Use the <code>--with-timestamp</code> option to easily keep more than one set of backup data under the same main backup directory.
<code>datadir</code> directory	A subdirectory that stores the data files and database subdirectories from the original MySQL instance.	Created under the backup directory by <code>mysqlbackup</code> .
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 <code>--use-tts</code> 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	Saved under the <code>datadir</code> directory inside the backup. A copy of the index file on the MySQL server that lists all the used binary log files, with the locations of the binary log files properly updated to point to the files' locations in the backup, is included into the backup as well also under the <code>datadir</code> directory. Use the <code>--skip-binlog</code> option to exclude the binary log from the backup.


File Name, Pattern, or Extension	Relation to Original Data Files	Notes
	a database to its state at a certain point in time after the last full backup. See Section 5.2, “Point-in-Time Recovery” for details.	<p>By default, the binary log files and the index file are restored to the same locations they were found on the backed-up server. Use the <code>--log-bin</code> option to specify a different target location for the binary log. Use the <code>--skip-binlog</code> option to skip the restoring of the binary log.</p> <p>The binary log files are compressed and saved with the <code>.bz</code> extension when being included in a compressed backup.</p> <div><div>Notes<ul style="list-style-type: none">• If any binary log files are missing on the server you are backing up, you should use the <code>--skip-binlog</code> option to avoid <code>mysqlbackup</code> throwing an error for the missing files.• No binary log files are copied into the incremental backup if the <code>--use-tts</code> option or the</div></div>


File Name, Pattern, or Extension	Relation to Original Data Files	Notes
		<p>-- start- lsn option is used. To include binary log files for the period 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.</p>


File Name, Pattern, or Extension	Relation to Original Data Files	Notes
		<ul style="list-style-type: none"> No binary log files are restored onto a server with a partial restore.
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 <code>--use-tts</code> option). Their inclusion saves the time and resources required for fetching the relay logs from the source when the replica is being restored.	<p>Saved under the <code>datadir</code> directory under the backup directory. A copy of the index file on the replica server that lists all the used relay log files, 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 as well, under the <code>datadir</code> directory. Use the <code>--skip-relaylog</code> option to exclude the relay log from the backup.</p> <p>By default, the relay log files and the index file are restored to the same locations they were found on the backed-up replica server. Use the <code>--relay-log</code> option to specify a different target location for the relay log. Use the <code>--skip-relaylog</code> option to skip the restoring of the relay log.</p> <p>No relay log files are restored onto a server with a partial restore.</p> <p>The relay log files are compressed and saved with the <code>.bz</code> extension when being included in a compressed backup.</p>
*.bz	Compressed binary log or relay log files.	The binary log and relay log files are compressed and saved with the <code>.bz</code> extension when being included in a compressed backup. They are decompressed during a restore.
undo log files	Undo log files from the server. See Undo Tablespaces for details.	Saved by default under the <code>datadir</code> directory inside the backup. Use the <code>--backup_innodb_undo_directory</code>


File Name, Pattern, or Extension	Relation to Original Data Files	Notes
	<p><i>For release 8.0.16 and later:</i> Both active and inactive undo tablespaces are included in the backup. Also, when the <code>--incremental-with-redo-log-only</code> option is used for creating incremental backups, <code>mysqlbackup</code> creates from the redo log an undo log for the period covered by the incremental backup, and includes it in the backup.</p>	<p>option to specify another location for the undo log in the backup.</p> <p><i>For release 8.0.15 and earlier:</i> The undo tablespaces are restored to the location pointed to by the <code>--innodb_undo_directory</code> option.</p> <p><i>For release 8.0.16 and later:</i> During a restore, the default undo tablespaces, as well as any non-default undo tablespaces resided in the backed-up server's data directory, are restored to the location pointed to by the <code>mysqlbackup</code> option <code>--innodb_undo_directory</code>. Non-default, external undo tablespaces are restored to the locations they were found on the backed-up server; change their restore locations by editing the <code>tablespace_tracker</code> file.</p> <p>No undo log files are restored onto a server with a partial restore.</p>
encrypted keyring data file	<p>For a server using the <code>keyring_encrypted_file</code> plugin, the file specified by the <code>keyring_encrypted_file_data</code> option on the server is copied over into the backup with its original name under the <code>meta</code> folder.</p> <p>For a server using the a keyring plugin other than <code>keyring_encrypted_file</code>, the file is named <code>keyring_kef</code>, saved under the <code>meta</code> folder.</p>	<p>An encrypted file containing the master key for InnoDB table encryption . See Chapter 6, Working with Encrypted InnoDB Tablespaces for detail.</p>
replica status log files	<p>Usually named <code>master.info</code> and <code>relay-log.info</code>, they are included by default in a backup of a replica database in a replication setup. See Replication Metadata Repositories, for details.</p>	<p>Saved under the <code>datadir</code> directory under the backup directory.</p> <p>The copying of these files are skipped during a backup or a restore when the <code>--skip-relay-log</code> option is used.</p>
Backup image file	<p>A single-file backup produced by the <code>backup-to-image</code> option, with a name specified by the <code>--backup-image</code> option.</p>	<p>You can move the image file without losing or damaging the contents inside it, then unpack it with <code>mysqlbackup</code> using</p>

File Name, Pattern, or Extension	Relation to Original Data Files	Notes
		the <code>extract</code> command and specifying the same image name with the <code>--backup-image</code> option. Although some extra files such as <code>backup-my.cnf</code> and the <code>meta</code> 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 <code>datadir</code> directory (that is, under <code>backup-dir/datadir/subdir</code>)	Copied from the database subdirectories under the MySQL data directory.	<p>By default, any unrecognized files in subdirectories under the MySQL data directory are copied to the backup. To omit such files, specify the <code>--only-known-file-types</code> option.</p> <div>  <div> Note <p>Some limitations apply to this behavior. See the discussion here in Appendix B, Limitations of MySQL Enterprise Backup.</p> </div> </div>
<code>meta</code> directory	A subdirectory that stores files with metadata about the backup.	Created under the backup directory by <code>mysqlbackup</code> . All files listed below go inside the <code>meta</code> subdirectory.
<code>backup_variables.txt</code>	Holds important information about the backup. For use by <code>mysqlbackup</code> only.	<code>mysqlbackup</code> consults and possibly updates this file during operations after the initial backup, such as the apply-log phase or the restore phase.
<code>image_files.xml</code>	Contains the list of all the files (except itself) that are present in the single-file backup produced by the <code>backup-to-image</code> or <code>backup-dir-to-image</code> options. For details about this file, see Section 16.4, “Using the MySQL Enterprise Backup Manifest” .	This file is not modified at any stage once generated.
<code>backup_create.xml</code>	Lists the command line arguments and environment in which the backup was created. For details about this file, see	This file is not modified once it is created. You can prevent this file from being generated

File Name, Pattern, or Extension	Relation to Original Data Files	Notes
	Section 16.4, “Using the MySQL Enterprise Backup Manifest” .	by specifying the <code>--disable-manifest</code> option.
<code>backup_content.xml</code>	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 16.4, “Using the MySQL Enterprise Backup Manifest” .	This file is not modified once created. You can prevent this file from being generated by specifying the <code>--disable-manifest</code> option.
<code>comments.txt</code>	Produced by the <code>--comments</code> or <code>--comments-file</code> option.	The comments are specified by you to document the purpose or special considerations for this backup job.
<code>backup_gtid_executed.sql</code>	Signifies the backup came from a server with GTIDs enabled.	<p>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 <code>mysqlbackup</code>, the file named <code>backup_gtid_executed.sql</code> is created in the <code>meta</code> folder under the backup directory. Edit and execute this file after restoring the backup data on a replica server; see Section 8.1, “Setting Up a New replica” for details.</p> <div>  <div> Note <p>For TTS backups for replica servers, use the <code>--slave-info</code> option to have <code>backup_gtid_executed.sql</code> generated.</p> </div> </div>
<code>server-my.cnf</code>	Contains values of the backed-up server's global variables that are set to non-default values. Use this file or <code>server-all.cnf</code> to start the target server for restoration.	During a <code>copy-back</code> or <code>copy-back-and-apply-log</code> operation, the <code>server repository options</code> values (e.g., <code>--datadir</code> , <code>--innodb_data_home_dir</code> ,

File Name, Pattern, or Extension	Relation to Original Data Files	Notes
		<p>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.</p> <div>  <div> Warning <p>When using the file to restart the target server, change parameters like <code>--tmpdir</code>, <code>--general-log</code>, etc., and any global variable that uses an absolute file path to avoid the accidental usage of the wrong file locations by the target server.</p> </div> </div>
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.	<p>During a copy-back or copy-back-and-apply-log operation, the server repository options values (e.g., <code>--datadir</code>, <code>--innodb_data_home_dir</code>, 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</p>

File Name, Pattern, or Extension	Relation to Original Data Files	Notes
		<p>saved in the file take precedence and they are not modified by the option values supplied through the command.</p> <div>  <div> Warning <p>When using the file to restart the target server, change parameters like <code>--tmpdir</code>, <code>--general-log</code>, etc., and any global variable that uses an absolute file path to avoid the accidental usage of the wrong file locations by the target server.</p> </div> </div>
<code>backup-auto.cnf</code>	Copy of the file <code>auto.cnf</code> from the backed-up server.	The file is restored into the data directory of the restored server. To use the UUID stored inside for your restored server, rename the file back to <code>auto.cnf</code> before you start the server.
<code>backup-mysqld-auto.cnf</code>	Copy of the file <code>mysqld-.cnf</code> from the backed-up server.	The file is restored into the data directory of the restored server. To use the persisted system variables stored inside for your restored server, rename the file back to <code>mysqld-auto.cnf</code> before you start the server.
<code>ib_buffer_pool</code>	The file produced on the server when <code>innodb_buffer_pool_dump_at_shutdown</code> (enabled by default on	With the default setting on MySQL server 5.7.7 and after <code>innodb_buffer_pool_load_at_startup</code> the target server, during start up,

File Name, Pattern, or Extension	Relation to Original Data Files	Notes
	<p>MySQL 5.7.7 and after) or <code>innodb_buffer_pool_dump_now</code> is enabled. It holds the list of tablespace IDs and page IDs of the server's buffer pool.</p> <p>The actual file name might be different, as it can be configured by the server's system variable <code>innodb_buffer_pool_filename</code>"</p>	<p>is going to restore the buffer pool state of the backed up server using this file. See Saving and Restoring the Buffer Pool State for details.</p>
<code>tablespace_tracker</code>	<p>The file tracks external tablespaces, recording their file paths on the backed-up server and their tablespace IDs.</p>	<p>If any external tablespace exists on a backed up server, the tracker file is going to be found in the <code>datadir</code> folder inside the backup. Change <code>server_file_path</code> in the file for any tablespace if you want to change the restore location for that tablespace (an absolute path must be used). To access the tracker file in a single-file backup, use the <code>extract</code> command.</p> <div>  <div> Warnings <ul style="list-style-type: none"> If the tracker file is deleted from the backup, the restore of the backup might fail silently, resulting in corruptions of the restored data. <i>For release 8.0.16 and later:</i> You cannot change the restore </div> </div>


File Name, Pattern, or Extension	Relation to Original Data Files	Notes
		<div>location for the default undo tablespaces of the database by editing their <code>server_file_path</code> entries. Their restore location is controlled by the setting of the <code>mysqlbackup</code> option</div> <div>-- <code>innodb_undo_dir</code></div> <div>After a restore is finished, if the restored server contains any external tablespace, a tracker file is going to be found in the data directory of the restored server.</div>

1.3.2 Files Backed up for InnoDB Data

The 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 (either as part of a `copy-back-and-apply-log` command or a `backup-and-apply-log` command, or as a separate `apply-log` command), 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, or for some other uses such as testing, reporting, or deployment as a replica.



Note

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

1.3.3 Files Backed up for Data Stored with MyISAM and Other Storage Engines

`mysqlbackup` also backs up the [.MYD files](#), [.MYI files](#), and the [.sdi files](#) associated with the MyISAM tables. Files with other extensions that are backed up are shown in [Table 1.1, “Types of Files in a Backup”](#).



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 be. They can be 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.

1.3.4 Files Generated by `mysqlbackup`

Inside the image backup file created by the `backup-to-image` command of `mysqlbackup` are 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.
- `*.bkt`: Transfer file created for an encrypted InnoDB table during backup. It contains the reencrypted tablespace key and other information related to the encryption. See [Chapter 6, Working with Encrypted InnoDB Tablespaces](#) for detail.

For details about these and other files contained in the backup, see [Table 1.1, “Types of Files in a Backup”](#).

1.4 The Backup Process

The following is a very brief outline of the steps performed by `mysqlbackup` when it creates a backup. It does not include every single step taken by `mysqlbackup`, and the description only represents a very general case—the process can look quite different, depending on the backup options you use (especially with some of the options described in [Section 19.10, “Performance / Scalability / Capacity Options”](#) and [Section 19.16, “Options for Special Backup Types”](#)). This description only applies to MySQL Enterprise Backup 8.0.16 and later.

In general, this is what happens when you run a backup operation with `mysqlbackup`:

1. The InnoDB data files, redo log, binary log, and relay log files (except for the log files currently in use) are being copied into the backup, while the database server operates as usual.

The data and the structures of the InnoDB tables might have changed during this period; so, some of the following steps are for making sure those changes are captured in the backup.

2. A [backup lock](#) is applied on the server instance. It blocks DDL operations (except for those on user-created temporary tables), but not DML operations (except for those not captured by the binary log, like administrative changes to the database) on InnoDB tables. Most read and write activities on the database are still allowed. With this lock applied, [mysqlbackup](#) scans for InnoDB tables that have been modified by DDL operations since step 1, and make changes to the backup accordingly.
3. A `FLUSH TABLES tbl_name [, tbl_name] ... WITH READ LOCK` statement is applied on all non-InnoDB tables (for release 8.0.18 and later, only on non-InnoDB tables that are to be included in the backup), after which any non-InnoDB tables relevant to the backup are copied.

This step is skipped if no user-created non-InnoDB tables exist on the database.

4. A brief blocking of logging activities on the server is applied, for [mysqlbackup](#) to collect logging-related information like the current InnoDB LSN, binary log position, GTID, replication source or replica status, and so on.
5. The read lock on the non-InnoDB tables is released.
6. Using information from step 4 above, the relevant portion of the binary or relay log file currently in use is copied. This ensures that all recent changes to the InnoDB tables since step 1 are captured in the backup, so they can be applied later to the [raw backup](#) data to bring the restored server to a consistent state.
7. The [backup lock](#) on the server instance is released. The database now returns to its normal operations.
8. The redo log files not yet copied before, as well as all the metadata files for the backup, are copied or created.
9. The backup operation is completed, and [mysqlbackup](#) returns success.

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.

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-8.0`. You must add this directory to your `$PATH` setting.

Installation packages for Debian and Ubuntu platforms are also available. Install the package with the following command `sudo dpkg -i package_name.deb`.

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 8.0 > 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 8.0?

This chapter highlights the new features in MySQL Enterprise Backup 8.0, as well as any significant changes made to MySQL Enterprise Backup with the release of this series.

- Offline backups are no longer supported. Using the old `--no-connection` and `--connect-if-online` options with MySQL Enterprise Backup will result in an error. The proper [connection options](#) must be supplied to MySQL Enterprise Backup when making a backup. The following options, used during offline backups, have also been removed :
 - `--innodb_data_file_path`
 - `--innodb_log_files_in_group`
 - `--innodb_log_file_size`
 - `--innodb_page_size`
 - `--innodb_checksum_algorithm`
 - `--keyring_file_data`
 - `--keyring_okv_conf_dir`
 - `--relay-log-info-file`
 - `--master-info-file`
 - `--backup_innodb_log_files_in_group`
 - `--backup_innodb_log_file_size`
 - `--backup_innodb_undo_tablespaces`
 - `--backup_innodb_undo_logs`
 - `--backup_innodb_page_size`
 - `--backup_innodb_checksum_algorithm`
- The binary log for a backed-up server, instead of being restored always to the data directory on the target server, is now restored by default to the same location it was found on the backed-up server. It can also be restored to a different location specified with the new `--log-bin` option.
- The relay log for a backed-up replica server, instead of being restored always to the data directory on the target replica server, is now restored by default to the same location it was found on the backed-up replica server. It can also be restored to a different location specified with the new `--relay-log` option.
- A file now tracks information of external tablespaces for a backup or restore in JSON format. See description for [tablespace_tracker](#) in [Table 1.1, "Types of Files in a Backup"](#) for details.
- A new option, `--tls-version`, specifies the protocols [mysqlbackup](#) permits for encrypted connections to MySQL servers.
- [MySQL Enterprise Firewall Overview](#) is now supported.
- The options `--ssl` and `--ssl-verify-server-cert`, already deprecated in MySQL Enterprise Backup 4.1, have now been removed. Use the `--ssl-mode` option instead to configure the security mode of your connection to the server.

-
- HTTP Basic Authentication and non-chunked transfer are now supported for backup and restore using OpenStack Swift-compatible object storage services. See [Section 19.15, “Cloud Storage Options”](#) for details.
 - The buffer size for cloud transfers can now be specified using the new `cloud-buffer-size` option. See [Section 19.15, “Cloud Storage Options”](#) for details.
 - Servers' use of the `keyring_encrypted_file` and the `keyring_aws` plugins is now supported. Also, irrespective of the keyring plugin type that is used on the server, the keyring data is now stored in the backup in an encrypted file. See [Chapter 6, Working with Encrypted InnoDB Tablespaces](#) for details.
 - The server option `--secure-auth`, deprecated since MySQL 5.7.5, is no longer supported by `mysqlbackup`.
 - The `backup_history` table now includes a `server_uuid` column, which stores the value of the `server_uuid` of the backed up server.
 - Due to the new features and functions of MySQL Enterprise Backup 8.0, more privileges are now required for the user by which `mysqlbackup` connects to the MySQL Server. See [Section 4.1.2, “Grant MySQL Privileges to Backup Administrator”](#) for details.
 - *For MySQL Enterprise Backup 8.0.12 and later:*
 - When working with a [Group Replication](#) setup, `mysqlbackup` now makes the backup history available to all members of the server group by making sure that the `backup_history` table is updated on a primary node after each `mysqlbackup` operation. See [Chapter 9, Using MySQL Enterprise Backup with Group Replication](#) for details, including the resulting new user privilege requirement for `mysqlbackup` to connect to a server, regardless of whether the server belongs to a Group Replication setup.
 - OAuth is now supported for Oracle Cloud Infrastructure Object Storage client authentication. Two new options, `--cloud-storage-url` and `--cloud-oauth-token`, have been introduced for the purpose. See [Section 19.15, “Cloud Storage Options”](#) for details.
 - The storage engine of the `mysql.backup_history` table on a backed-up server has switched from CSV to InnoDB. See [here](#) for the special user privileges now required by `mysqlbackup` that are related to this change.
 - *For MySQL Enterprise Backup 8.0.13 and later:*
 - `mysqlbackup` now supports [transparent page compression](#) for InnoDB tables. The support is enabled by setting the `mysqlbackup` option `--compress-method=punch-hole`; see description for the option for details.
 - `mysqlbackup` now supports backup compression (i.e., the use of the `--compress` and `--uncompress` options) for incremental backups (except for incremental backups created with the `--incremental-with-redo-log-only` option).
 - *For MySQL Enterprise Backup 8.0.14 and later:*
 - `mysqlbackup` now supports the `--ssl-fips-mode` option, which controls whether `mysqlbackup` operates in FIPS mode. See [FIPS Support](#) for details.
 - `mysqlbackup` now supports encrypted binary and relay log. See [Section 8.4, “Working with Encrypted Binary and Relay Logs”](#) for details.

- For MySQL Enterprise Backup 8.0.16 and later:

- Near the end of the backup process, instead of locking the whole server instance for a brief period of time, `mysqlbackup` now applies these locks consecutively:
 1. A [backup lock](#) on the server instance, which blocks DDLs (except those on user-created temporary tables), but not DMLs on InnoDB tables.
 2. A `FLUSH TABLES tbl_name [, tbl_name] ... WITH READ LOCK` operation on all non-InnoDB tables, for copying the relevant ones among them into the backup. This step is skipped if no user created non-InnoDB tables exist.
 3. A brief blocking of logging activities on the server, for collecting logging-related information.

See [Section 1.4, “The Backup Process”](#) for details. The removal of the lock on the whole server instance reduces disruption to the database service by the backup operation.



Important

The change requires that the [BACKUP_ADMIN](#) and [SELECT](#) privileges on all tables be granted to the user by which `mysqlbackup` connects to the server (the [BACKUP_ADMIN](#) privilege is automatically granted to users with the [RELOAD](#) privilege when an in-place upgrade to MySQL Server 8.0 from an earlier version is performed).

- In addition to the requirement that the target data directory for a restore specified by `--datadir` must be non-existent or empty, `mysqlbackup` now enforces the same rule for the `--innodb_data_home_dir`, `--innodb_log_group_home_dir`, and `--innodb_undo_directory` options (the `--force` option cannot be used to override the requirement on the three options).
- `mysqlbackup` now supports [dynamic changes to undo tablespaces on the server being backed up](#). During a restore, the default undo tablespaces, as well as any non-default undo tablespaces resided in the backed-up server's data directory, are restored to the location pointed to by the `mysqlbackup` option `innodb_undo_directory`. Non-default, external undo tablespaces are restored to the locations they were found on the backed-up server. See [descriptions for the undo log files](#) for details.
- `mysqlbackup` now supports [encrypted InnoDB undo logs](#). The encrypted undo tablespaces are handled the same way as the encrypted tablespaces for InnoDB tables. See [Chapter 6, Working with Encrypted InnoDB Tablespaces](#) for details.
- For MySQL Enterprise Backup 8.0.17 and later:
 - `mysqlbackup` now supports [encrypted InnoDB redo logs](#). The encrypted redo tablespaces are handled the same way as the encrypted tablespaces for InnoDB tables. See [Chapter 6, Working with Encrypted InnoDB Tablespaces](#) for details.
 - To avoid `mysqlbackup` failing to catch up with the growing redo log during a backup operation and missing redo log data, `mysqlbackup` now utilizes [redo log archiving](#), a new feature available on MySQL Server 8.0.17. Redo log archiving can be skipped using the new `mysqlbackup` option `--no-redo-log-archive`. See [Chapter 7, Backing up Using Redo Log Archiving](#) for details.
 - The `--incremental-base` option now accepts a new value, `history:last_full_backup`, which makes it easy to create a [differential backup](#). See the description of `--incremental-base` for details.

- For MySQL Enterprise Backup 8.0.18 and later:
 - `mysqlbackup` now supports a faster way to create incremental backups by using the page tracking functionality on MySQL Servers. To use this new feature, set `--incremental=page-track`. See [Incremental Backup Using Page Tracking](#) for details.
 - The `--uncompress` option is now supported for the `extract` operation, so that files from a compressed single-file backup can now be extracted and uncompressed with a single command.
 - Two new options, `--compression-algorithms` and `--zstd-compression-level`, have been introduced for configuring [compression for server connections](#). See [Section 19.2, “Connection Options”](#) and [Command Options for Connection Compression](#) for details.
 - The `image-to-backup-dir` command is now an alias for the `extract` command.
- For MySQL Enterprise Backup 8.0.19 and later:
 - It is now safe to have DDL operations ([CREATE TABLE](#), [RENAME TABLE](#), [DROP TABLE](#), [ALTER TABLE](#), and operations that map to [ALTER TABLE](#) like [CREATE INDEX](#)) happening on the server in parallel with a backup operation; see details described [here](#) for some limitations. Due to this new feature, the user by which `mysqlbackup` connects to the MySQL server must now be granted the `SELECT ON *.*` privilege; see [Section 4.1.2, “Grant MySQL Privileges to Backup Administrator”](#).
 - [Master key rotation for binary log encryption](#) on the server in between a full and an incremental backup, as well as between two incremental backups performed by `mysqlbackup`, is now supported. During an incremental backup, `mysqlbackup` now records encryption information for all the encrypted binary log files (including those already backed up in earlier full or incremental backups) unless the `--skip-binlog` option is used, in which case a warning is given that the older binary log files might become unrestoreable.

Also, the `--skip-binlog` option now makes binary log to be skipped not just for the current backup operation, but also for all subsequent incremental backups that are based on the current backup.

- The logging for backup restore has been improved: at the steps for setting the sizes of the log files, the names of the log files are now included.
- `mysqlbackup` now prints a stack trace after being terminated by a signal.
- When `mysqlbackup` fails to connect to a server, the warning returned by `mysqlbackup` now includes the hostname and port number for TCP connections, and the socket information for socket connections. This is particularly helpful for a Group Replication setup, for which `mysqlbackup` might attempt to connect to more than one host.
- The storage engine for the `mysql.backup_progress` table on a backed-up server has switched from CSV to InnoDB, and a composite index is now created on the table's `backup_id`, `current_timestamp` columns. Also, when working with a Group Replication setup, `mysqlbackup` now makes the `backup_progress` table available to all members of the server group by making sure that the table is updated on a primary node during each `mysqlbackup` operation.

When MySQL Enterprise Backup 8.0.19 or later tries to perform its first full backup on a database, it automatically checks the format of the `mysql.backup_progress` table. If it detects that the table is in the old format (which means the server has been upgraded from 8.0.18 or earlier and has been backed up by MySQL Enterprise Backup before), it attempts to perform an update on the table automatically. New privileges for the `mysqlbackup` user on the server are required for the table upgrade; see [Appendix F, Backup Progress Table Update](#) for details.

- `mysqlbackup` now includes the configuration files `auto.cnf` and `mysqld-auto.cnf` from a server in its backup (except for a TTS backup). They are restored to the target server's data

directory as `backup-auto.cnf` and `backup-mysqld-auto.cnf` respectively. To use those files to configure your restored server, rename them to their original names before starting the server.

- For the `backup-to-image`, `extract`, `list-image`, and `copy-back-and-apply-log` commands, any *relative path* specified with `--backup-image` is now taken to be relative to the current directory in which the command is run.
- *For MySQL Enterprise Backup 8.0.20 and later:*
 - The `tablespace_tracker` file has been simplified: it now contains only two fields for each external tablespace: `server_file_path` and `space_id`. `mysqlbackup` no longer relies on the file for information on the `backup_file_path` and the tablespace type, which means that users no longer need to update the `tablespace_tracker` file when they move a directory backup to a new location.
 - Table-Level Recovery (TLR) now allows selective restores of tables or schemas from full backups; see [Section 5.1.4, “Table-Level Recovery \(TLR\)”](#) for details.
 - The legacy option `--include` is now deprecated. A deprecation warning is now issued by `mysqlbackup` whenever the option is used. The `--include-tables` and `--exclude-tables` options should be used instead for partial backups and restores.
- *For MySQL Enterprise Backup 8.0.21 and later:*
 - For a `backup-to-image` operation, when a relative path is specified for the `--backup-image` option, `mysqlbackup` interprets the file path given as relative to the `backup directory`.
 - Encrypted InnoDB tables can now be included in partial backups and restores using [transportable tablespaces \(TTS\)](#).
 - Encrypted InnoDB tables are now being verified in `validate` operations.
 - Compressed InnoDB files are now being verified in `validate`, `backup`, and `backup-to-image` operations.
 - The storage engine for the `mysql.backup_sbt_history` table on a backed-up server has switched from CSV to InnoDB. Also, an auto-increment primary key `id` column has been added to the table. When working with a Group Replication setup, `mysqlbackup` now makes the `backup_sbt_history` table available to all members of the server group by making sure that the table is updated on a primary node during each `mysqlbackup` operation.

When MySQL Enterprise Backup 8.0.21 or later tries to perform its first full backup on a database using the SBT API, it automatically checks the format of the `mysql.backup_sbt_history` table. If it detects that the table is in the old format (which means the server has been upgraded from 8.0.20 or earlier and has been backed up by MySQL Enterprise Backup before using the SBT API), it attempts to perform an update on the table automatically. New privileges for the `mysqlbackup` user on the server are required for the table upgrade; see [Appendix E, SBT Backup History Table Update](#) for details.

- Commands for operations on compressed backups (`copy-back`, `copy-back-and-apply-log`, `apply-log`, etc.) have been simplified: the `--uncompress` option is no longer needed, except for `extract` and `image-to-backup-dir` operations that *do not* use the `--src-entry` option.
- Commands for operations on incremental backups (`copy-back`, `copy-back-and-apply-log`, `apply-log`) have been simplified: the `--incremental` option is no longer needed for those operations.

-
- A backup now fails when a binary or relay log file is purged while the backup is going on; it also fails when `mysqlbackup` finds a binary log file missing on the server (however, if a relay log file is missing, the backup continues).
 - The `tool_name` column of the `backup_progress` table on the MySQL server is now populated with the full `mysqlbackup` command that invoked a backup operation.
 - The file `backup_gtid_executed.sql` was not included in a TTS backup for a replica server using GTIDs. The file is now included in a TTS backup as long as the `--slave-info` option is used.

Part II Using MySQL Enterprise Backup

Table of Contents

4	Backing Up a Database Server	35
4.1	Before the First Backup	35
4.1.1	Collect Database Information	35
4.1.2	Grant MySQL Privileges to Backup Administrator	36
4.1.3	Designate a Location for the Backup Directory	39
4.2	The Typical Backup / Verify / Restore Cycle	40
4.2.1	Backing Up an Entire MySQL Instance	40
4.2.2	Verifying a Backup	42
4.2.3	Restoring a Database	43
4.3	Backup Scenarios and Examples	46
4.3.1	Making a Single-File Backup	46
4.3.2	Making a Full Backup	50
4.3.3	Making a Differential or Incremental Backup	51
4.3.4	Making a Compressed Backup	57
4.3.5	Making a Partial Backup	57
4.3.6	Making an Optimistic Backup	60
4.3.7	Making a Back Up of In-Memory Database Data	63
4.3.8	Making Scheduled Backups	63
4.4	Making Backups with a Distributed File System (DFS) or Storage Access Network (SAN) ..	64
5	Recovering or Restoring a Database	65
5.1	Performing a Restore Operation	65
5.1.1	Restoring a Compressed Backup	66
5.1.2	Restoring an Encrypted Backup Image	67
5.1.3	Restoring an Incremental Backup	67
5.1.4	Table-Level Recovery (TLR)	68
5.1.5	Restoring Backups Created with the <code>--use-tts</code> Option	69
5.1.6	Restoring a Backup from Cloud Storage to a MySQL Server	69
5.1.7	Restoring External InnoDB Tablespaces to Different Locations	70
5.1.8	Advanced: Preparing and Restoring a Directory Backup	71
5.2	Point-in-Time Recovery	71
5.3	Restoring a Backup with a Database Upgrade or Downgrade	73
6	Working with Encrypted InnoDB Tablespaces	75
7	Backing up Using Redo Log Archiving	79
8	Using MySQL Enterprise Backup with Replication	81
8.1	Setting Up a New replica	81
8.2	Backing up and Restoring a Replica Database	83
8.3	Restoring a Source Database	83
8.4	Working with Encrypted Binary and Relay Logs	85
9	Using MySQL Enterprise Backup with Group Replication	87
10	Encryption for Backups	89
11	Using MySQL Enterprise Backup with Media Management Software (MMS) Products	91
11.1	Backing Up to Tape with Oracle Secure Backup	91
12	Using MySQL Enterprise Backup with Docker	95
13	Performance Considerations for MySQL Enterprise Backup	97
13.1	Optimizing Backup Performance	97
13.2	Optimizing Restore Performance	100
14	Monitoring Backups with MySQL Enterprise Monitor	103
15	Using MySQL Enterprise Backup with MySQL Enterprise Firewall	105
16	Troubleshooting for MySQL Enterprise Backup	107
16.1	Exit codes of MySQL Enterprise Backup	107
16.2	Working Around Corruption Problems	108
16.3	Using the MySQL Enterprise Backup Logs	109
16.4	Using the MySQL Enterprise Backup Manifest	110

Chapter 4 Backing Up a Database Server

Table of Contents

4.1 Before the First Backup	35
4.1.1 Collect Database Information	35
4.1.2 Grant MySQL Privileges to Backup Administrator	36
4.1.3 Designate a Location for the Backup Directory	39
4.2 The Typical Backup / Verify / Restore Cycle	40
4.2.1 Backing Up an Entire MySQL Instance	40
4.2.2 Verifying a Backup	42
4.2.3 Restoring a Database	43
4.3 Backup Scenarios and Examples	46
4.3.1 Making a Single-File Backup	46
4.3.2 Making a Full Backup	50
4.3.3 Making a Differential or Incremental Backup	51
4.3.4 Making a Compressed Backup	57
4.3.5 Making a Partial Backup	57
4.3.6 Making an Optimistic Backup	60
4.3.7 Making a Back Up of In-Memory Database Data	63
4.3.8 Making Scheduled Backups	63
4.4 Making Backups with a Distributed File System (DFS) or Storage Access Network (SAN)	64

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 to Use It
Path to MySQL configuration file	Default system locations, hardcoded application default locations, or from the <code>--defaults-file</code> option in the <code>mysqld</code> startup script.	The preferred way to convey database configuration information to mysqlbackup is to use the <code>--defaults-file</code> option. When connection and data layout information is available from the configuration file, you no longer need to supply separately most of the information listed below.
MySQL port	MySQL configuration file or mysqld startup script.	Used to connect to the database instance during backup operations. Specified via the <code>--</code>

Information to Gather	Where to Find It	How to Use It
		<code>port</code> option of <code>mysqlbackup</code> . The specification is not needed if the information is available from the MySQL configuration file.
Path to MySQL data directory	MySQL configuration file or <code>mysqld</code> 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.
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.	Specified via the <code>--password</code> option of the <code>mysqlbackup</code> . Prompted at the terminal if the <code>--password</code> option is present without the password argument.
Path under which to store backup data or metadata, temporarily or permanently	You choose this. See Section 4.1.3, “Designate a Location for the Backup Directory” for details.	In general, this directory has to be empty for <code>mysqlbackup</code> to write data into it.
Owner and permission information for backed-up files (for Linux, Unix, and OS X systems)	In the MySQL data directory.	If you do the backup using a different OS user ID or a different <code>umask</code> setting than applies to the original files, you might need to run commands such as <code>chown</code> and <code>chmod</code> on the backup data. See Appendix B, Limitations of MySQL Enterprise Backup for details.
Size of InnoDB redo log files	Calculated from the values of the <code>innodb_log_file_size</code> and <code>innodb_log_files_in_group</code> configuration variables. Use the technique explained for the <code>--incremental-with-redo-log-only</code> option.	Only needed if you perform incremental backups using the <code>--incremental-with-redo-log-only</code> option rather than the <code>--incremental</code> 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 <code>--incremental-with-redo-log-only</code> option.	Only needed if you perform incremental backups using the <code>--incremental-with-redo-log-only</code> option rather than the <code>--incremental</code> 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`. Here are the privileges required by `mysqlbackup`:

- The minimum privileges for the MySQL user with which `mysqlbackup` connects to the server include:
 - *For MySQL Enterprise Backup 8.0.19 and later:* `SELECT` on all databases and tables, for table locks that protect the backups against inconsistency caused by parallel DDL operations.
 - *For MySQL Enterprise Backup 8.0.16 and later:*
 - `BACKUP_ADMIN` on all databases and tables.
 - `SELECT` on `performance_schema.variables_info` and `performance_schema.log_status`.
 - `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.
 - `PROCESS`, to process DDL statements with the `ALGORITHM = INPLACE` clause.
 - `SELECT` on `performance_schema.replication_group_members`, to know whether the server instance is part of a Group Replication setup and, if so, to gather information on the group members (required by release 8.0.12 and later).

To create a MySQL user (`mysqlbackup` in this example) and set the above-mentioned privileges for the user to connect from localhost, issue statements like the following from the `mysql` client program:

```
CREATE USER 'mysqlbackup'@'localhost' IDENTIFIED BY 'password';
GRANT SELECT ON *.* TO 'mysqlbackup'@'localhost'; # For release 8.0.19 and later
GRANT BACKUP_ADMIN ON *.* TO 'mysqlbackup'@'localhost'; # For release 8.0.16 and later
GRANT SELECT ON performance_schema.variables_info TO 'mysqlbackup'@'localhost'; # For release 8.0.16
GRANT SELECT ON performance_schema.log_status TO 'mysqlbackup'@'localhost'; # For release 8.0.16 to 8
GRANT RELOAD ON *.* TO 'mysqlbackup'@'localhost';
GRANT CREATE, INSERT, DROP, UPDATE ON mysql.backup_progress TO 'mysqlbackup'@'localhost';
GRANT CREATE, INSERT, DROP, UPDATE, SELECT, ALTER ON mysql.backup_history TO 'mysqlbackup'@'localhost';
GRANT REPLICATION CLIENT ON *.* TO 'mysqlbackup'@'localhost';
GRANT SUPER ON *.* TO 'mysqlbackup'@'localhost';
GRANT PROCESS ON *.* TO 'mysqlbackup'@'localhost';
GRANT SELECT ON performance_schema.replication_group_members TO 'mysqlbackup'@'localhost'; # For release
```

- The following additional privileges are required **when using MySQL Enterprise Backup 8.0.19 or later for the first time on a MySQL Server that has been upgraded from 8.0.18 or earlier and has been backed up by MySQL Enterprise Backup before**:
 - `ALTER` on `mysql.backup_progress`.
 - `CREATE`, `INSERT`, and `DROP` on `mysql.backup_progress_old`.
 - `CREATE`, `INSERT`, `DROP`, and `ALTER` on `mysql.backup_progress_new`.

Grant these privileges by issuing these sample statements at the `mysql` client:

```
GRANT ALTER ON mysql.backup_progress TO 'mysqlbackup'@'localhost';
GRANT CREATE, INSERT, DROP ON mysql.backup_progress_old TO 'mysqlbackup'@'localhost';
GRANT CREATE, INSERT, DROP, ALTER ON mysql.backup_progress_new TO 'mysqlbackup'@'localhost';
```

**Note**

If you are working with a multiprimary Group Replication setting, make sure these privileges are granted on all primary nodes; see also [Chapter 9, Using MySQL Enterprise Backup with Group Replication](#).

These privileges are for the attempt to migrate the `mysql.backup_progress` table to a newer format (see [Appendix F, Backup Progress Table Update](#) for details), and they are no longer needed after the first backup operation by MySQL Enterprise Backup 8.0.19 or later has taken place on the server, by which point they can be revoked.

- The following additional privileges are required **when using MySQL Enterprise Backup 8.0.12 or later for the first time on a MySQL Server that has been upgraded from 8.0.11 or earlier and has been backed up by MySQL Enterprise Backup before**:

- `CREATE`, `INSERT`, and `DROP` on `mysql.backup_history_old`.
- `CREATE`, `INSERT`, `DROP`, and `ALTER` on `mysql.backup_history_new`.

Grant these privileges by issuing these sample statements at the `mysql` client:

```
GRANT CREATE, INSERT, DROP ON mysql.backup_history_old TO 'mysqlbackup'@'localhost';
GRANT CREATE, INSERT, DROP, ALTER ON mysql.backup_history_new TO 'mysqlbackup'@'localhost';
```

**Note**

If you are working with a multiprimary Group Replication setting, make sure these privileges are granted on all primary nodes; see also [Chapter 9, Using MySQL Enterprise Backup with Group Replication](#).

These privileges are for the attempt to migrate the `mysql.backup_history` table to a newer format (see [Appendix D, Backup History Table Update](#) for details), and they are no longer needed after the first backup operation by MySQL Enterprise Backup 8.0.12 or later has taken place on the server, by which point they can be revoked.

- The following additional privileges are required for using specific features of MySQL Enterprise Backup:
 - 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 directory.
 - For [creating tape backups using the System Backup to Tape \(SBT\) API](#) :
 - `CREATE`, `INSERT`, `DROP`, and `UPDATE` on the `mysql.backup_sbt_history` table
 - For [working with encrypted InnoDB tables](#):
 - `ENCRYPTION_KEY_ADMIN` to enable InnoDB encryption key rotation.
 - For [MySQL Enterprise Backup 8.0.16 and later](#): For backing up and restoring user-created non-InnoDB tables:
 - `SELECT` on all user-created non-InnoDB tables
 - `LOCK TABLES` on all schemas containing user-created non-InnoDB tables

- For MySQL Enterprise Backup 8.0.17 and later: For using [redo log archiving](#) for backups:
 - `INNODB_REDO_LOG_ARCHIVE` to invoke the user-defined function `innodb_redo_log_archive_start()`.

Set those additional privileges if you are using the features that require them. To set all of them, issue statements like the following from the `mysql` client program:

```
GRANT LOCK TABLES, SELECT, CREATE, DROP, FILE ON *.* TO 'mysqlbackup'@'localhost';
GRANT CREATE, INSERT, DROP, UPDATE ON mysql.backup_sbt_history TO 'mysqlbackup'@'localhost';
GRANT ENCRYPTION_KEY_ADMIN ON *.* TO 'mysqlbackup'@'localhost';
GRANT SELECT ON non-InnoDB_tbl TO 'mysqlbackup'@'localhost'; # For release 8.0.16 and later
GRANT INNODB_REDO_LOG_ARCHIVE ON *.* TO 'mysqlbackup'@'localhost'; # For release 8.0.17 and later
```

- The following additional privileges are required **when performing for the first time a backup using the SBT API with MySQL Enterprise Backup 8.0.21 or later on a MySQL Server that has been upgraded from 8.0.20 or earlier and has been backed up by MySQL Enterprise Backup before using the SBT API**:
 - `ALTER` on `mysql.backup_sbt_history`.
 - `CREATE`, `INSERT`, and `DROP` on `mysql.backup_sbt_history_old`.
 - `CREATE`, `INSERT`, `DROP`, and `ALTER` on `mysql.backup_sbt_history_new`.

Grant these privileges by issuing these sample statements at the `mysql` client:

```
GRANT ALTER ON mysql.backup_sbt_history TO 'mysqlbackup'@'localhost';
GRANT CREATE, INSERT, DROP ON mysql.backup_sbt_history_old TO 'mysqlbackup'@'localhost';
GRANT CREATE, INSERT, DROP, ALTER ON mysql.backup_sbt_history_new TO 'mysqlbackup'@'localhost';
```



Note

If you are working with a multiprimary Group Replication setting, make sure these privileges are granted on all primary nodes; see also [Chapter 9, Using MySQL Enterprise Backup with Group Replication](#).

These privileges are for the attempt to migrate the `mysql.backup_sbt_history` table to a newer format (see [Appendix E, SBT Backup History Table Update](#) for details), and they are no longer needed after the first backup operation by MySQL Enterprise Backup 8.0.21 or later using the SBT API has taken place on the server, by which point they can be revoked.

- For privileges required for using MySQL Enterprise Backup with a Group Replication setting, see [Chapter 9, Using MySQL Enterprise Backup with Group Replication](#).

4.1.3 Designate a Location for the Backup Directory

Most `mysqlbackup` operations, including those on single-file backups, write data or metadata to a designated directory referred to as the “[backup directory](#)” in this manual. See the description for `--backup-dir` for details on its usage for different operations.

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

If you use the backup directory as a location to store your backups, 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`.

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 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 `mysqlbackup` 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=mysqlbackup --password --host=127.0.0.1 --backup-image=/home/mysqlbackup/backups/my.mbi
--backup-dir=/home/mysqlbackup/backup-tmp backup-to-image
MySQL Enterprise Backup Ver 8.0.16-commercial for linux-glibc2.12 on x86_64 (MySQL Enterprise - Commercial
Copyright (c) 2003, 2019, Oracle and/or its affiliates. All rights reserved.

Oracle is a registered trademark of Oracle Corporation and/or its
affiliates. Other names may be trademarks of their respective
owners.

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

IMPORTANT: Please check that mysqlbackup run completes successfully.
At the end of a successful 'backup-to-image' run mysqlbackup
prints "mysqlbackup completed OK!".

190402 12:56:04 MAIN      INFO: Starting to log actions.
Enter password:
190402 12:56:09 MAIN      INFO: No SSL options specified.
190402 12:56:09 MAIN      INFO: MySQL server version is '8.0.16-commercial'
190402 12:56:09 MAIN      INFO: MySQL server compile os version is 'linux-glibc2.12'
190402 12:56:09 MAIN      INFO: Got some server configuration information from running server.

190402 12:56:09 MAIN      INFO: Backup directory exists: '/home/mysqlbackup/backup-tmp'
190402 12:56:09 MAIN      INFO: MySQL server version_comment is 'MySQL Enterprise Server - Commercial'
190402 12:56:09 MAIN      INFO: Server is not a community server.
190402 12:56:09 MAIN      INFO: KEF target path: '/home/mysqlbackup/backup-tmp/meta/keyring_kef'
190402 12:56:09 MAIN      INFO: TDE Keyring service initialized.
190402 12:56:09 MAIN      INFO: MEB logfile created at /home/mysqlbackup/backup-tmp/meta/MEB_2019-04-02.12-56

-----
Server Repository Options:
-----
datadir                = /home/admin/bin/mysql-commercial-8.0.16/datadir/
innodb_data_home_dir   =
innodb_data_file_path  = ibdata1:12M:autoextend
innodb_log_group_home_dir = /home/admin/bin/mysql-commercial-8.0.16/datadir/
innodb_log_files_in_group = 2
innodb_log_file_size   = 50331648
innodb_undo_directory  = /home/admin/bin/mysql-commercial-8.0.16/datadir/
innodb_undo_tablespace = 2
innodb_buffer_pool_filename = ib_buffer_pool
innodb_page_size       = 16384
innodb_checksum_algorithm = crc32
```

```

-----
Backup Config Options:
-----
datadir                = /home/mysqlbackup/backup-tmp/datadir
innodb_data_home_dir   = /home/mysqlbackup/backup-tmp/datadir
innodb_data_file_path   = ibdata1:12M:autoextend
innodb_log_group_home_dir = /home/mysqlbackup/backup-tmp/datadir
innodb_log_files_in_group = 2
innodb_log_file_size    = 50331648
innodb_undo_directory   = /home/mysqlbackup/backup-tmp/datadir
innodb_undo_tablespace  = 2
innodb_buffer_pool_filename = ib_buffer_pool
innodb_page_size        = 16384
innodb_checksum_algorithm = crc32

Backup Image Path = /home/mysqlbackup/backups/my.mbi
190402 12:56:09 MAIN      INFO: Unique generated backup id for this is 15542241696070511

190402 12:56:09 MAIN      INFO: Creating 14 buffers each of size 16777216.
190402 12:56:09 MAIN      INFO: Full Image Backup operation starts with following threads
      1 read-threads      6 process-threads      1 write-threads
190402 12:56:09 MAIN      INFO: Found checkpoint at lsn 19840686.
190402 12:56:09 MAIN      INFO: Starting log scan from lsn = 19840512 at offset = 32256 and checkpoint =
190402 12:56:09 RDR1      INFO: Copying meta file /home/mysqlbackup/backup-tmp/backup-my.cnf.
190402 12:56:09 RDR1      INFO: Copying meta file /home/mysqlbackup/backup-tmp/meta/backup_create.xml.
190402 12:56:09 RDR1      INFO: Starting to copy all innodb files...
190402 12:56:09 RDR1      INFO: Copying /home/admin/bin/mysql-commercial-8.0.16/datadir/ibdata1.
190402 12:56:09 RDR1      INFO: Starting to copy all undo files...
190402 12:56:09 RDR1      INFO: Copying /home/admin/bin/mysql-commercial-8.0.16/datadir/undo_002.
190402 12:56:09 RDR1      INFO: Copying /home/admin/bin/mysql-commercial-8.0.16/datadir/undo_001.
190402 12:56:09 RDR1      INFO: Copying /home/admin/bin/mysql-commercial-8.0.16/datadir/sys/sys_config.ib
190402 12:56:09 RDR1      INFO: Copying /home/admin/bin/mysql-commercial-8.0.16/datadir/pets/cats.ibd.
190402 12:56:09 RDR1      INFO: Copying /home/admin/bin/mysql-commercial-8.0.16/datadir/mysql/backup_hist
190402 12:56:09 RDR1      INFO: Copying /home/admin/bin/mysql-commercial-8.0.16/datadir/mysql.ibd.
190402 12:56:10 RDR1      INFO: Completing the copy of innodb files.
190402 12:56:10 RDR1      INFO: Requesting a dump of the InnoDB buffer pool
190402 12:56:10 RDR1      INFO: Waiting for the dump of the InnoDB buffer pool to complete
190402 12:56:10 RDR1      INFO: The dump of the InnoDB buffer pool completed
190402 12:56:10 RDR1      INFO: Binary Log Basename: '/home/admin/bin/mysql-commercial-8.0.16/datadir/bin
190402 12:56:10 RDR1      INFO: Binary Log Index:    '/home/admin/bin/mysql-commercial-8.0.16/datadir/bin
190402 12:56:10 RDR1      INFO: Relay Channel:        'group_replication_applier'
190402 12:56:10 RDR1      INFO: Relay Log Basename: '/home/admin/bin/mysql-commercial-8.0.16/datadir/admi
190402 12:56:10 RDR1      INFO: Relay Log Index:      '/home/admin/bin/mysql-commercial-8.0.16/datadir/admi
190402 12:56:10 RDR1      INFO: Relay Channel:        'group_replication_recovery'
190402 12:56:10 RDR1      INFO: Relay Log Basename: '/home/admin/bin/mysql-commercial-8.0.16/datadir/admi
190402 12:56:10 RDR1      INFO: Relay Log Index:      '/home/admin/bin/mysql-commercial-8.0.16/datadir/admi
190402 12:56:10 RDR1      INFO: Starting to copy Binlog files...
190402 12:56:10 RDR1      INFO: Copying /home/admin/bin/mysql-commercial-8.0.16/datadir/binlog.000001.
190402 12:56:10 RDR1      INFO: Copying /home/admin/bin/mysql-commercial-8.0.16/datadir/binlog.000002.
190402 12:56:10 RDR1      INFO: Copying /home/admin/bin/mysql-commercial-8.0.16/datadir/binlog.000003.
190402 12:56:10 RDR1      INFO: Starting to lock instance for backup...
190402 12:56:10 RDR1      INFO: No SSL options specified.
190402 12:56:10 RDR1      INFO: The server instance is locked for backup.
190402 12:56:10 RDR1      INFO: Starting to read-lock tables...
190402 12:56:10 RDR1      INFO: No tables to read-lock.
190402 12:56:10 RDR1      INFO: Opening backup source directory '/home/admin/bin/mysql-commercial-8.0.16/
190402 12:56:10 RDR1      INFO: Starting to copy non-innodb files in subdirs of '/home/admin/bin/mysql-co
190402 12:56:10 WTR1      INFO: Adding database directory: datadir/mysql
190402 12:56:10 WTR1      INFO: Adding database directory: datadir/performance_schema
190402 12:56:10 RDR1      INFO: Completing the copy of all non-innodb files.
190402 12:56:10 WTR1      INFO: Adding database directory: datadir/pets
190402 12:56:10 WTR1      INFO: Adding database directory: datadir/sys
190402 12:56:10 RDR1      INFO: Requesting consistency information...
190402 12:56:10 RDR1      INFO: Locked the consistency point for 3089 microseconds.
190402 12:56:10 RDR1      INFO: Consistency point server_uuid 'a01aa59f-5567-11e9-ad69-08002759ceb5'.
190402 12:56:10 RDR1      INFO: Consistency point gtid_executed ''.
190402 12:56:10 RDR1      INFO: Consistency point binary_log_file 'binlog.000004'.
190402 12:56:10 RDR1      INFO: Consistency point binary_log_position 379.
190402 12:56:10 RDR1      INFO: Consistency point InnoDB lsn 19840686.
190402 12:56:10 RDR1      INFO: Consistency point InnoDB lsn_checkpoint 19840686.
190402 12:56:10 RDR1      INFO: Requesting completion of redo log copy after LSN 19840686.
190402 12:56:10 RLR1      INFO: Redo log reader waited = 0.00 ms for logs to generate.

```

```

190402 12:56:10 RLW1 INFO: A copied database page was modified at 19840686. (This is the highest lsn fou
190402 12:56:10 RLW1 INFO: Scanned log up to lsn 19840686.
190402 12:56:10 RDR1 INFO: Copying /home/admin/bin/mysql-commercial-8.0.16/datadir/binlog.000004.
190402 12:56:10 RDR1 INFO: Completed the copy of binlog files...
190402 12:56:10 RDR1 INFO: The server instance is unlocked after 0.085 seconds.
190402 12:56:10 RDR1 INFO: Reading all global variables from the server.
190402 12:56:10 RDR1 INFO: Completed reading of all 548 global variables from the server.
190402 12:56:10 RDR1 INFO: Writing server defaults files 'server-my.cnf' and 'server-all.cnf' for server
190402 12:56:10 RDR1 INFO: Copying meta file /home/mysqlbackup/backup-tmp/meta/backup_variables.txt.
190402 12:56:10 RDR1 INFO: Copying meta file /home/mysqlbackup/backup-tmp/datadir/ibbackup_logfile.
190402 12:56:10 RDR1 INFO: Copying meta file /home/mysqlbackup/backup-tmp/server-all.cnf.
190402 12:56:10 RDR1 INFO: Copying meta file /home/mysqlbackup/backup-tmp/server-my.cnf.
190402 12:56:10 RDR1 INFO: Copying meta file /home/mysqlbackup/backup-tmp/meta/backup_content.xml.
190402 12:56:10 RDR1 INFO: Copying meta file /home/mysqlbackup/backup-tmp/meta/image_files.xml.
190402 12:56:10 MAIN INFO: Full Image Backup operation completed successfully.
190402 12:56:10 MAIN INFO: Backup image created successfully.
190402 12:56:10 MAIN INFO: Image Path = /home/mysqlbackup/backups/my.mbi
190402 12:56:10 MAIN INFO: MySQL binlog position: filename binlog.000004, position 379

-----
Parameters Summary
-----
Start LSN          : 19840512
End LSN            : 19840686
-----

mysqlbackup completed OK!

```

4.2.2 Verifying a Backup

You can check the integrity of your backup using the `validate` command. The following is a sample command for validating a backup image and the output for the successful validation:

```

$ mysqlbackup --backup-image=/home/mysqlbackup/backups/my.mbi validate
MySQL Enterprise Backup Ver 8.0.16-commercial for linux-glibc2.12 on x86_64 (MySQL Enterprise - Commercial
Copyright (c) 2003, 2019, Oracle and/or its affiliates. All rights reserved.

Oracle is a registered trademark of Oracle Corporation and/or its
affiliates. Other names may be trademarks of their respective
owners.

Starting with following command line ...
bin/mysqlbackup --backup-image=/home/mysqlbackup/backups/my.mbi validate

IMPORTANT: Please check that mysqlbackup run completes successfully.
At the end of a successful 'validate' run mysqlbackup
prints "mysqlbackup completed OK!".

190401 17:27:14 MAIN INFO: Starting to log actions.
190401 17:27:14 MAIN INFO: Backup Image MEB version string: 8.0.16 [2019-03-26 18:51:33]
190401 17:27:14 MAIN INFO: MySQL server version is '8.0.16'
190401 17:27:14 MAIN INFO: TDE Keyring service initialized.
190401 17:27:14 MAIN INFO: Creating 14 buffers each of size 16777216.
190401 17:27:14 MAIN INFO: Validate operation starts with following threads
    1 read-threads    6 process-threads
190401 17:27:14 MAIN INFO: Validating image ... /home/mysqlbackup/backups/my.mbi
190401 17:27:14 PCR1 INFO: Validate: [Dir]: meta
190401 17:27:14 PCR1 INFO: Validate: [Dir]: datadir/mysql
190401 17:27:14 PCR1 INFO: Validate: [Dir]: datadir/performance_schema
190401 17:27:14 PCR1 INFO: Validate: [Dir]: datadir/pets
190401 17:27:14 PCR1 INFO: Validate: [Dir]: datadir/sys
190401 17:27:14 MAIN INFO: Total files as specified in image: 137
190401 17:27:14 MAIN INFO: datadir/ibdata1 validated.
190401 17:27:14 MAIN INFO: datadir/undo_002 validated.
190401 17:27:14 MAIN INFO: datadir/undo_001 validated.
190401 17:27:14 MAIN INFO: datadir/sys/sys_config.ibd validated.
190401 17:27:14 MAIN INFO: datadir/pets/cats.ibd validated.
190401 17:27:14 MAIN INFO: datadir/mysql/backup_history.ibd validated.
190401 17:27:14 MAIN INFO: datadir/mysql.ibd validated.
190401 17:27:14 MAIN INFO: Validate operation completed successfully.
190401 17:27:14 MAIN INFO: Backup Image validation successful.

```

```
190401 17:27:14 MAIN      INFO: Source Image Path = /home/mysqlbackup/backups/my.mbi
mysqlbackup completed OK!
```

Furthermore, you can also verify that your backup has been successful by restoring 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 4.2.3, “Restoring a Database”](#) for the basic steps for restoring a backup, and see [Chapter 5, *Recovering or Restoring a Database*](#) for more detailed instructions.



Warning

Do not try to verify a backup by using the [backup directory](#) as a data directory to start a MySQL server. This will crash the server, and might also corrupt your backup. See [Appendix A, *Frequently Asked Questions for MySQL Enterprise Backup*](#) for details.

4.2.3 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 18.3, “Restore Operations”](#).

In the example below, the single-file backup created in the example given in [Section 4.2.1, “Backing Up an Entire MySQL Instance”](#) is restored using the `copy-back-and-apply-log` command. The following options are used:

- `--datadir` supplies the location of the data directory for restoring the data. You must specify this option for any restore operation, either at the command line or in a defaults file.
- `--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 --datadir=/home/admin/bin/mysql-commercial-8.0.16/datadir \
  --backup-image=/home/mysqlbackup/backups/my.mbi --backup-dir=/home/mysqlbackup/backup-tmp \
  copy-back-and-apply-log
MySQL Enterprise Backup Ver 8.0.16-commercial for linux-glibc2.12 on x86_64 (MySQL Enterprise - Commer
Copyright (c) 2003, 2019, Oracle and/or its affiliates. All rights reserved.

Oracle is a registered trademark of Oracle Corporation and/or its
affiliates. Other names may be trademarks of their respective
owners.

Starting with following command line ...
mysqlbackup --datadir=/home/admin/bin/mysql-commercial-8.0.16/datadir
  --backup-image=/home/mysqlbackup/backups/my.mbi
  --backup-dir=/home/mysqlbackup/backup-tmp copy-back-and-apply-log

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!".

190402 16:56:37 MAIN      INFO: Starting to log actions.
```

```

190402 16:56:37 MAIN      INFO: Backup directory exists: '/home/mysqlbackup/backup-tmp'
190402 16:56:37 MAIN      INFO: Backup Image MEB version string: 8.0.16 [2019-03-26 18:51:33]
190402 16:56:37 MAIN      INFO: MySQL server version is '8.0.16'
190402 16:56:37 MAIN WARNING: If you restore to a server of a different version, the innodb_data_file_path
190402 16:56:37 MAIN WARNING: If you restore to a server of a different version, the innodb_log_files_in_g
190402 16:56:37 MAIN WARNING: If you restore to a server of a different version, the innodb_log_file_size p
190402 16:56:37 MAIN      INFO: Server is not a community server.
190402 16:56:37 MAIN      INFO: KEF source path: '/home/mysqlbackup/backup-tmp/meta/keyring_kef'
190402 16:56:37 MAIN      INFO: KEF target path: '/home/admin/bin/mysql-commercial-8.0.16/datadir/keyring_kef'
190402 16:56:37 MAIN      INFO: TDE Keyring service initialized.
190402 16:56:37 MAIN      INFO: MEB logfile created at /home/mysqlbackup/backup-tmp/meta/MEB_2019-04-02.16-56

-----
                        Server Repository Options:
-----
datadir                = /home/admin/bin/mysql-commercial-8.0.16/datadir
innodb_data_home_dir    = /home/admin/bin/mysql-commercial-8.0.16/datadir
innodb_data_file_path   = ibdata1:12M:autoextend
innodb_log_group_home_dir = /home/admin/bin/mysql-commercial-8.0.16/datadir
innodb_log_files_in_group = 2
innodb_log_file_size    = 50331648
innodb_undo_directory   = /home/admin/bin/mysql-commercial-8.0.16/datadir
innodb_undo_tablespaces = 2
innodb_buffer_pool_filename = ib_buffer_pool
innodb_page_size        = Null
innodb_checksum_algorithm = crc32

-----
                        Backup Config Options:
-----
datadir                = /home/mysqlbackup/backup-tmp/datadir
innodb_data_home_dir    = /home/mysqlbackup/backup-tmp/datadir
innodb_data_file_path   = ibdata1:12M:autoextend
innodb_log_group_home_dir = /home/mysqlbackup/backup-tmp/datadir
innodb_log_files_in_group = 2
innodb_log_file_size    = 50331648
innodb_undo_directory   = /home/mysqlbackup/backup-tmp/datadir
innodb_undo_tablespaces = 2
innodb_buffer_pool_filename = ib_buffer_pool
innodb_page_size        = 16384
innodb_checksum_algorithm = crc32

190402 16:56:37 MAIN      INFO: Creating 14 buffers each of size 16777216.
190402 16:56:37 MAIN      INFO: Copy-back-and-apply-log operation starts with following threads
   1 read-threads   6 process-threads   1 write-threads
190402 16:56:37 PCR1      INFO: Copying database directory: meta
190402 16:56:37 RDR1      INFO: Copying ibdata1.
190402 16:56:37 RDR1      INFO: Copying undo_002.
190402 16:56:37 RDR1      INFO: Copying undo_001.
190402 16:56:37 RDR1      INFO: Copying sys/sys_config.ibd.
190402 16:56:37 RDR1      INFO: Copying pets/cats.ibd.
190402 16:56:37 RDR1      INFO: Copying mysql/backup_history.ibd.
190402 16:56:37 RDR1      INFO: Copying mysql.ibd.
190402 16:56:37 RDR1      INFO: Copying binlog.000001.
190402 16:56:37 RDR1      INFO: Copying binlog.000002.
190402 16:56:37 RDR1      INFO: Copying binlog.000003.
190402 16:56:37 PCR3      INFO: Copying database directory: mysql
190402 16:56:37 PCR3      INFO: Copying database directory: performance_schema
190402 16:56:37 PCR3      INFO: Copying database directory: pets
190402 16:56:37 PCR3      INFO: Copying database directory: sys
190402 16:56:37 RDR1      INFO: Copying binlog.000004.
190402 16:56:37 MAIN      INFO: Total files as specified in image: 138
190402 16:56:37 MAIN      INFO: MySQL server version is '8.0.16-commercial'
190402 16:56:37 MAIN      INFO: Restoring ...8.0.16-commercial version
190402 16:56:37 MAIN      INFO: Copy-back operation completed successfully.
190402 16:56:37 MAIN      INFO: Source Image Path = /home/mysqlbackup/backups/my.mbi

190402 16:56:37 MAIN      INFO: MySQL server version is '8.0.16-commercial'
190402 16:56:37 MAIN      INFO: Restoring ...8.0.16-commercial version
190402 16:56:37 MAIN      INFO: Creating 14 buffers each of size 65536.
190402 16:56:37 MAIN      INFO: Apply-log operation starts with following threads

```

```

1 read-threads      1 process-threads      6 apply-threads
190402 16:56:37 MAIN      INFO: Using up to 100 MB of memory.
190402 16:56:37 MAIN      INFO: ibbackup_logfile's creation parameters:
        start lsn 19841024, end lsn 19841405,
        start checkpoint 19841405.
190402 16:56:37 MAIN      INFO: Loading the space id : 0, space name : /home/admin/bin/mysql-commercial-8
190402 16:56:37 MAIN      INFO: Apply log: Loading the undo space id : 4294967278, space name : /home/adm
190402 16:56:37 MAIN      INFO: Apply log: Loading the undo space id : 4294967279, space name : /home/adm
190402 16:56:37 MAIN      INFO: Loading the space id : 3, space name : /home/admin/bin/mysql-commercial-8
190402 16:56:37 MAIN      INFO: Loading the space id : 2, space name : /home/admin/bin/mysql-commercial-8
190402 16:56:37 MAIN      INFO: Loading the space id : 1, space name : /home/admin/bin/mysql-commercial-8
190402 16:56:37 MAIN      INFO: Loading the space id : 4294967294, space name : /home/admin/bin/mysql-com
190402 16:56:37 PCRL      INFO: Starting to parse redo log at lsn = 19841394, whereas checkpoint_lsn = 19
190402 16:56:37 PCRL      INFO: Doing recovery: scanned up to log sequence number 19841405.
190402 16:56:37 PCRL      INFO: Starting to apply a batch of log records to the database....
InnoDB: Progress in percent:
190402 16:56:37 PCRL      INFO: Updating last checkpoint to 19841405 in redo log/
190402 16:56:37 PCRL      INFO: Setting log file size to 50331648
190402 16:56:37 PCRL      INFO: Setting log file size to 50331648
190402 16:56:37 PCRL      INFO: Log file header:
        format = 3
        pad1 = 0
        start lsn = 19841024
        checkpoint lsn = 19841405
        checksum = 2051460933
        creator = MEB 8.0.16
190402 16:56:37 PCRL      INFO: We were able to parse ibbackup_logfile up to
        lsn 19841405.
190402 16:56:37 PCRL      INFO: Last MySQL binlog file position 0 379, file name binlog.000004
190402 16:56:37 PCRL      INFO: The first data file is '/home/admin/bin/mysql-commercial-8.0.16/datadir/i
        and the new created log files are at '/home/admin/bin/mysql-commercial-8.
190402 16:56:37 MAIN      INFO: No Keyring file to process.
190402 16:56:37 MAIN      INFO: Apply-log operation completed successfully.
190402 16:56:37 MAIN      INFO: Full Backup has been restored successfully.

mysqlbackup completed OK! with 3 warnings

```

Now the original database directory is restored from the backup.

Starting the Restored Server. When the following InnoDB settings are different on the backed-up and the restored server, it is important to configure the restored server with the settings from the backed up server (otherwise, your restored server might not start):

- `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 backed-up server, they were actually 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 restored the single-image backup, or in a backup directory you could 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 using to start the target server afterwards; alternatively, you can also supply them as command line options to `mysqld`.

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 5.1, “Performing a Restore Operation”](#).

4.3 Backup Scenarios and Examples

4.3.1 Making a Single-File Backup

To avoid having a large number of backup files to keep track, store, and transport, `mysqlbackup` conveniently creates backups in a single-file format. It can also pack an existing backup directory into a single file, unpack the single file back to a backup directory, 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 19.9, “Single-File Backup Options”](#).

Advanced: While `mysqlbackup` can also create a directory backup (see description for the `backup` command for details) instead of a single-file backup, the single-file format is preferable in most cases: a single-file backup is easier to handle and store, and certain functions of `mysqlbackup` are not supported for directory backups—for example, backup to cloud and backup to tape using the System Backup to Tape (SBT) API. Throughout the manual, directory backup is mostly treated as an advanced topic, and information and examples for directory backups are marked with the *Advanced* tag, like this paragraph.

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 4.1 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 4.2 Single-File Backup to Relative Path

When a relative path instead of an absolute path was supplied with the `--backup-image` option, the path is taken to be relative to the [backup directory](#). Therefore, in this example, the resulting single-file backup is created as `/backups/sales.mbi`.

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

Example 4.3 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 4.4 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 4.5 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 4.6 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 4.7 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 4.8 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 4.9 Extract Single-File Backup into a Backup Directory

This command extracts all contents of a single-file backup into the directory specified with the `--backup-dir` option.

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

Example 4.10 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 4.11 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. (Notice the slash (/) at the end of the value `meta/` for `--src-entry`, without which all files or folders containing the string `meta` in their pathnames will be extracted.)

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

Example 4.12 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.

4.3.1.1 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 `copy-back-and-apply-log` command on the other end to restore the backup to a remote MySQL server.

Example 4.13 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 4.14 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=. --uncompress --backup-image=- copy-back-and-apply-log'
```

Example 4.15 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=- co
```

4.3.1.2 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 11, Using MySQL Enterprise Backup with Media Management Software \(MMS\) Products](#).

4.3.1.3 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.

MySQL Enterprise Backup 8.0 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.

A cloud backup is created using the cloud options for `mysqlbackup`, which are described in details in [Section 19.15, “Cloud Storage Options”](#). Here are some sample commands for creating a cloud backup:

Example 4.16 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 4.17 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 4.18 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 4.19 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
```

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 4.20 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=<serviceName>--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 4.21 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 5.1.6, “Restoring a Backup from Cloud Storage to a MySQL Server”](#) on how to restore a backup image from a cloud storage.

4.3.2 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 4.2.1, “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 20, Configuration Files and Parameters](#) for details about the use of configuration files.

Using a Single Backup Directory or Timestamped Subdirectories?

For convenience, the `--with-timestamp` option creates uniquely named subdirectories under the [backup directory](#) to hold the backup data (permanent or temporary) and metadata. 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), specify a new, unique directory name for each backup job.

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 4.3.3, “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, 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. For more discussions on compressed backups, see [Section 4.3.4, “Making a Compressed Backup”](#).

4.3.3 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; you should also ignore any instructions that only apply to the handling of multiple incremental backups.

**Note**

For MySQL Enterprise Backup 8.0.17 and later, you can create a differential backup easily using the option `--incremental-base=history:last_full_backup`.

See [Section 19.7, “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 is. 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 log data 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.
- Backup compression (i.e., use of the [compression options](#)) is not supported when you perform incremental backups with the redo log only. If backup compression is important to you, do not use the `--incremental-with-redo-log-only` option.

Incremental Backup Using Page Tracking

For MySQL Enterprise Backup 8.0.18 and later: `mysqlbackup` supports creating incremental backups using the page tracking functionality of the MySQL Server, by which `mysqlbackup` looks for changed

pages in the InnoDB data files that have been modified since the last backup and then copies them. In general, incremental backups using page tracking are faster than other kinds of incremental backups performed by `mysqlbackup` if the majority of the data in the database has not been modified. Using this feature requires the following to be done on the server before the base backup for the incremental backup is made:

- Install the `mysqlbackup` component, which comes with the MySQL Enterprise Server 8.0 installation, by running this command at a `mysql` client connected to the server:

```
INSTALL COMPONENT "file://component_mysqlbackup";
```

- Start page tracking with the following user-defined function (UDF):

```
SELECT mysqlbackup_page_track_set(true);
```

The LSN value starting from which changed pages have been tracked is returned by this UDF:

```
SELECT mysqlbackup_page_track_get_start_lsn();
```

You can stop page tracking with the following UDF:

```
SELECT mysqlbackup_page_track_set(false);
```



Note

The above-mentioned UDFs regarding page tracking require the `BACKUP_ADMIN` privilege to run.

When the `--incremental` option is used without any value specified, `mysqlbackup` performs an incremental backup using the page tracking functionality. User can also specify `--incremental=page-track` to make `mysqlbackup` use the page tracking functionality. However, the prerequisites for making use of the page tracking functionality for incremental backups are:

- Page tracking is functioning properly on the server, and it has been enabled (with `SELECT mysqlbackup_page_track_set(true)`) before the base backup was created; if that is not the case, `mysqlbackup` throws an error when `--incremental=page-track`, or it performs a full-scan incremental backup instead when `--incremental` is unspecified.
- The number of changed pages is less than 50% of the total number of pages; if that is not the case, `mysqlbackup` throws an error when `--incremental=page-track`, or it performs a full-scan incremental backup instead when `--incremental` is unspecified.



Note

`mysqlbackup` needs to be started with enough memory to process all the tracked pages in memory. If there is not enough memory, `mysqlbackup` throws an error and then exits. A rough guideline is as follows: the default value of 300 [MB] for the `--limit-memory` option allows `mysqlbackup` to handle about 600GB of changed data.

Full-scan versus Optimistic Incremental Backup

When the `--incremental` option is set to `full-scan`, `mysqlbackup` performs a full-scan incremental backup, in which it scans all InnoDB data files in the server's data directory to find pages that have been changed since the last backup was made and then copies those pages. A full-scan incremental backup might not be very efficient when not many tables have been modified since the last backup.

An optimistic incremental backup, on the other hand, only scans for changed pages in InnoDB data files that have been modified since the last backup, thus saving some unnecessary scan time. An optimistic incremental backup can be performed by specifying `--incremental=optimistic`. While an optimistic incremental backup might shorten the backup time, it has the following limitations:

- Since this feature makes use of the modification times of the files in the server's data directory, two things must have remained unchanged since the previous backup: (1) the system time on the server, and (2) the location of the data directory. Otherwise, the backup might fail, or an inconsistent incremental backup might be produced.
- Optimistic incremental backups cannot be performed with the `--incremental-with-redo-log-only`, for which `mysqlbackup` reads the redo log files instead of scanning the files in the data directory.
- If the `--start-lsn` option is used, a full scan is performed even if `--incremental=optimistic` is specified since, in that case, `mysqlbackup` cannot determine the point in time for which the previous backup is consistent, and thus has no time frame to determine which files have been modified recently.

For these and other cases in which an optimistic incremental backup is not desirable, perform a full-scan incremental backup, or an incremental backup using page tracking (for MySQL Enterprise Backup 8.0.18 and later). See [Section 4.1.2, “Grant MySQL Privileges to Backup Administrator”](#) on the privileges required for `mysqlbackup` to perform an optimistic incremental backup. Also see [Using Optimistic Backups and Optimistic Incremental Backups Together](#) on how to utilize the two features together in a backup schedule.

For MySQL Enterprise Backup 8.0.17 and earlier, full-scan backup is the default method for incremental backups, which is utilized if no value is specified for `--incremental`.

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.

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

These examples use `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 do one of the following:

- Tell `mysqlbackup` to query the `end_lsn` value from the last successful [non-TTS backup](#) as recorded in the `backup_history` table on the server using `--incremental-base=history:last_backup` or `history:last_full_backup` (for release 8.0.17 and later).
- *Advanced:* For directory backups, specify the directory of the previous backup (either full or incremental) with `--incremental-base=dir:directory_path`, and `mysqlbackup` will figure

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. If your last backup was a single-file, you can still use `--incremental-base=dir:directory_path` to provide the location of the temporary directory you supplied with the `--backup-dir` option during the last backup

In the following example, the `--incremental-base=history:last_backup` option is used, given which `mysqlbackup` fetches the LSN of the last successful (non-TTS) full or partial backup from the `mysql.backup_history` table and performs an incremental backup basing on that.

```
mysqlbackup --defaults-file=/home/dbadmin/my.cnf \
--incremental --incremental-base=history:last_backup \
--backup-dir=/home/dbadmin/temp_dir \
--backup-image=incremental_image1.bi \
backup-to-image
```

In the following example, an incremental backup similar to the one in the last example but *optimistic* in nature is performed.

```
mysqlbackup --defaults-file=/home/dbadmin/my.cnf \
--incremental=optimistic --incremental-base=history:last_backup \
--backup-dir=/home/dbadmin/temp_dir \
--backup-image=incremental_image1.bi
backup-to-image
```

Advanced: Use the following command to create an incremental directory backup using the `--incremental-base=dir:directory_path` option; the backup is saved at the location specified by `--incremental-backup-dir`:

```
mysqlbackup --defaults-file=/home/dbadmin/my.cnf --incremental \
--incremental-base=dir:/incr-backup/wednesday \
--incremental-backup-dir=/incr-backup/thursday \
backup
```

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.

To create an incremental backup image with the `--start-lsn` option, use the following command, specifying with `--backup-dir` the backup directory, which, in this case, is a 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
```

Maintaining a backup schedule:

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

On how to restore your database using the incremental backups, see [Section 5.1.3, “Restoring an Incremental Backup”](#)

4.3.4 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 [Creating Compressed Tables](#)); 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 cannot use the `--compress` option for incremental backups created only with the redo log (i.e., with the `--incremental-with-redo-log-only` option).

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 19.6, “Compression Options”](#) for details.

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

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

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

```
mysqlbackup --defaults-file=/etc/my.cnf --compress --compress-level=5 backup
```

This is a sample command for making a compressed and [prepared](#) directory backup:

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

4.3.5 Making a Partial Backup

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 19.8, “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.

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=$MEB_TEMP_BACKUP_DIR --backup-image=$MEB_BACKUPS_DIR/my.mbi \
--include-tables=".emp" \
backup-to-image
```

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=$MEB_TEMP_BACKUP_DIR --backup-image=$MEB_BACKUPS_DIR/my.mbi \
--exclude-tables="^(mysql|performance_schema)\." \
backup-to-image
```

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=$MEB_TEMP_BACKUP_DIR --backup-image=$MEB_BACKUPS_DIR/my.mbi \
--include-tables="^sales\." --exclude-tables="^sales\.hardware$" \
backup-to-image
```

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):

```
mysqlbackup \
--host=localhost --user=mysqluser --protocol=TCP --port=3306 \
--backup-dir=$MEB_TEMP_BACKUP_DIR --backup-image=$MEB_BACKUPS_DIR/my.mbi \
```

```
--include-tables="^sales reps\." --exclude-tables="^sales reps\.euro-asia" \
backup-to-image
```

Backing up all InnoDB tables:

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

You can also make [compressed](#) and other kinds of selective backups by using the appropriate command options.

Making a Partial Backup with the Legacy Options



Important

Information in this subsection is only for using the legacy option of `--include`, which has been deprecated. For creating partial backups, use the `--include-tables` and `--exclude-tables` options instead.



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: 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.

Example 4.22 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. Of these 10 tables six (`alex1`, `alex2`, `alex3`, `blobt3`, `ibstest0`, `ibstest09`) are stored in per-table data files (`.ibd` files).

```
$ ls /sqldata/mts/test
alex2.ibd  ibstest0.ibd  alex1.ibd  blobt3.ibd  alex3.ibd  ibtest09.ibd
```

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

```
# Back up some InnoDB tables.
$ mysqlbackup --defaults-file=/home/dbadmin/my.cnf --include="^test\.ib.*"

# Contents in the backup directory's subdirectory for the test database:
$ ls /sqldata-backup/test
ibstest0.ibd  ibtest09.ibd
```

The backup directory's subdirectory for the `test` database contains only backups of `ibstest0` and `ibtest09` tables, because other InnoDB tables do not match the include pattern `^test\.ib.*`.

Example 4.23 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
alex2.ibd  ibstest0.ibd  alex1.ibd  blobt3.ibd  alex3.ibd  ibtest09.ibd
```

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

```
$ mysqlbackup --defaults-file=/home/dbadmin/my.cnf --compress \
--include=".*\.(alex|blob).*" 
```

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
```

4.3.6 Making an 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.

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 any locks on 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 various locks applied to the database at different times. 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 19.10, “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 4.24 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-image=<image-name> --backup-dir=<temp-dir> backup-to-image
```

Example 4.25 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-to-image
```

Example 4.26 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 4.27 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
```

Using Optimistic Backups and Optimistic Incremental Backups Together

By utilizing [optimistic backup](#) and [optimistic incremental backup](#) together in your backup schedule, you can speed up backups for huge databases, especially when only a relatively small number of tables have been modified since a certain time and not many tables are being modified on a frequent basis. Below is a sample sequence of commands illustrating a weekly backup schedule that makes use of the two features; it also includes the steps for restoring the data to a certain day.

```
# A full optimistic backup performed on 2017/02/04, Sat, at 1130 PM.
# The --optimistic-time option is used to specify an optimistic time of 2011/05/16, 0800 PM

mysqlbackup --defaults-file=/home/admin/my.cnf --optimistic-time=110516200000 \
  --backup-dir=/home/admin/temp_dir --backup-image=/home/admin/backups/mydb_full_201702042330.bi \
  --with-timestamp \
  backup-to-image

# A sequence of optimistic incremental backups are then performed on each the following six days at 1130 PM
# On Sunday, 2017/02/05
mysqlbackup --defaults-file=/home/admin/my.cnf \
  --incremental=optimistic --incremental-base=history:last_backup \
  --backup-dir=/home/admin/temp_dir \
  --backup-image=/home/admin/backups/mydb_incremental__201702052330.bi \
  --with-timestamp \
  backup-to-image
# On Monday, 2017/02/06
mysqlbackup --defaults-file=/home/admin/my.cnf \
  --incremental=optimistic --incremental-base=history:last_backup \
  --backup-dir=/home/admin/temp_dir \
  --backup-image=/home/admin/backups/mydb_incremental__201702062330.bi \
  --with-timestamp \
  backup-to-image
# On Tuesday, 2017/02/07
mysqlbackup --defaults-file=/home/admin/my.cnf \
  --incremental=optimistic --incremental-base=history:last_backup \
  --backup-dir=/home/admin/temp_dir \
  --backup-image=/home/admin/backups/mydb_incremental__201702072330.bi \
  --with-timestamp \
  backup-to-image
# On Wednesday, 2017/02/08
mysqlbackup --defaults-file=/home/admin/my.cnf \
  --incremental=optimistic --incremental-base=history:last_backup \
  --backup-dir=/home/admin/temp_dir \
  --backup-image=/home/admin/backups/mydb_incremental__201702082330.bi \
  --with-timestamp backup-to-image
# On Thursday, 2017/02/09
mysqlbackup --defaults-file=/home/admin/my.cnf \
  --incremental=optimistic --incremental-base=history:last_backup \
  --backup-dir=/home/admin/temp_dir \
  --backup-image=/home/admin/backups/mydb_incremental__201702092330.bi \
  --with-timestamp \
  backup-to-image
# On Friday, 2017/02/10
mysqlbackup --defaults-file=/home/admin/my.cnf \
  --incremental=optimistic --incremental-base=history:last_backup \
  --backup-dir=/home/admin/temp_dir \
  --backup-image=/home/admin/backups/mydb_incremental__201702102330.bi \
  --with-timestamp \
  backup-to-image

# Another full optimistic backup is performed on Saturday, 2017/02/11
mysqlbackup --defaults-file=/etc/my.cnf --optimistic-time=110516200000 \
  --backup-dir=/home/admin/temp_dir --backup-image=/home/admin/backups/mydb_full_201702112330.bi \
  --with-timestamp \
  backup-to-image

# Restore the database to its state at Tuesday, 2017/02/07, at 11:30 PM
# First, restore the full optimistic backup taken on the Saturday before, which was 2017/02/04:
mysqlbackup --defaults-file=/etc/my.cnf --backup-image=/home/admin/backups/mydb_full_201702042330.bi \
  --backup-dir=/home/admin/temp_dir --datadir=/var/lib/mysql \
```



```
--with-timestamp \
copy-back-and-apply-log
# Next, restore the optimistic incremental taken on the Sunday, Monday, and Tuesday that follow:
mysqlbackup --defaults-file=/etc/my.cnf --backup-image=/home/admin/backups/mydb_incremental__2017020523
--backup-dir=/home/admin/temp_dir --datadir=/var/lib/mysql --incremental \
--with-timestamp \
copy-back-and-apply-log
mysqlbackup --defaults-file=/etc/my.cnf --backup-image=/home/admin/backups/mydb_incremental__2017020623
--backup-dir=/home/admin/temp_dir --datadir=/var/lib/mysql --incremental \
--with-timestamp \
copy-back-and-apply-log
mysqlbackup --defaults-file=/etc/my.cnf --backup-image=/home/admin/backups/mydb_incremental__2017020723
--backup-dir=/home/admin/temp_dir --datadir=/var/lib/mysql --incremental \
--with-timestamp \
copy-back-and-apply-log
```

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's non-InnoDB tables are 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:

```
shell> crontab -e
```

- In the editor, add an entry similar to the following one to the crontab, and then save your changes (make sure contents in both lines below appear in a single line in the crontab):

```
@daily /path-to-mysqlbackup/mysqlbackup -uroot --backup-dir=/path-to-backup-folder/cronbackups
--with-timestamp --backup-image=my.mib backup-to-image &>/dev/null
```

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-timestamp --backup-image=my.mib backup-to-image &>/dev/null
```

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, especially when [using the redo log archiving feature](#) available on MySQL Server 8.0.17 . 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 Performing a Restore Operation	65
5.1.1 Restoring a Compressed Backup	66
5.1.2 Restoring an Encrypted Backup Image	67
5.1.3 Restoring an Incremental Backup	67
5.1.4 Table-Level Recovery (TLR)	68
5.1.5 Restoring Backups Created with the <code>--use-tts</code> Option	69
5.1.6 Restoring a Backup from Cloud Storage to a MySQL Server	69
5.1.7 Restoring External InnoDB Tablespaces to Different Locations	70
5.1.8 Advanced: Preparing and Restoring a Directory Backup	71
5.2 Point-in-Time Recovery	71
5.3 Restoring a Backup with a Database Upgrade or Downgrade	73

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 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 5.1.8, “Advanced: Preparing and Restoring a Directory Backup”](#)). 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 a [partial restore](#). 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 5.1 Restoring a Database

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

The `copy-back-and-apply-log` command achieves two things:

- Extracts the backup from the image file and copies it to the data directory on the server to be restored.
- Performs an [apply log operation](#) to the restored data to bring them up-to-date.

See [Section 4.2.3, “Restoring a Database”](#) for an explanation of the important options used in a restore operation like `--defaults-file`, `--datadir`, `--backup-image`, and `--backup-dir`.

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.



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 partial restores, for which other requirements described in [Section 5.1.4, “Table-Level Recovery \(TLR\)”](#) apply.
- After a full restore, 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
```

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

5.1.1 Restoring a Compressed Backup



Note

For MySQL Enterprise Backup 8.0.21 and later: The `--uncompress` option is no longer needed when restoring a compressed backup.

Restore a compressed backup image named `<image_name>`, using the `--backup-dir` option to specify the temporary directory into which temporary output, status files, and backup metadata will be saved:

Example 5.2 Restoring a Compressed Backup

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

Advanced: Do the same for a compressed directory backup at `<backupDir>` to `<restoreDir>` on the server using `copy-back-and-apply-log`:

Example 5.3 Restoring a Compressed Directory 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 4.0.1 and later), use the `copy-back` command and the `--uncompress` option:

Example 5.4 Restoring a Compressed and Prepared Directory Backup

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

See [Section 4.3.4, “Making a Compressed Backup”](#) and [Section 19.6, “Compression Options”](#) for more details on compressed backups.

5.1.2 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 5.5 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-
```

See [Section 19.13, “Encryption Options”](#) for more details on backup encryption and decryption.

5.1.3 Restoring an Incremental Backup



Note

For MySQL Enterprise Backup 8.0.21 and later: The `--incremental` option is no longer needed when 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 5.1, “Restoring a Database”](#) 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 5.6 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.

Advanced: Restoring an Incremental Backup Directory

Incremental directory backups can be restored in a series of `copy-back-and-apply-log` command, as illustrated above for single-file backups. Alternatively, at anytime after an incremental backup is taken and before the data is restored, 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 using the `apply-incremental-backup` command:

```
$ 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.

Binary Log and Relay Log Restore

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.

Location of the binary log (or relay log) after an incremental backup is restored is, by default, the same as the log's location on the backed-up server when the incremental backup was taken, or as specified by the `--log-bin` (or `--relay-log`) option during the restore of the incremental backup.

See [Section 4.3.3, “Making a Differential or Incremental Backup”](#), and [Section 19.7, “Incremental Backup Options”](#), for more details on incremental backups.

5.1.4 Table-Level Recovery (TLR)

For MySQL Enterprise Backup 8.0.20 and later: Table-Level Recovery (TLR) allows selected tables (or schemas) to be restored from a backup (be it a full backup, a partial backup, or a backup created using [transportable tablespaces \(TTS\)](#)) using the `--include-tables` and `--exclude-tables` options. The feature is also known as *partial restore* in this manual. Here are some general requirements for performing a TLR or partial restore:

- The destination server must be running.
- 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.
- *For non-TTS backups:* The tables being restored must already exist on the destination server, in the same table definition.
- *For TTS backups:* The tables being restored must *not* already exist on the destination server.
- While it is *not* necessary to specify the `--datadir` option when partially restoring a backup, *if* the option is specified, its value must match that of the target server, or the restore operation will fail (when restoring a TTS backup with release 8.0.16 or earlier, the `--datadir` option is required).

Here are some limitations for a TLR or partial restore:

- Individual partitions cannot be selectively restored. Tables selected by the `--include-tables` and `--exclude-tables` options are always restored in full.
- Partial restores cannot be performed with incremental backups.
- Binary, relay, and undo logs are not restored.
- *For non-TTS backups only*, these additional limitations apply:
 - After partial restores, tables could contain changes from uncommitted transactions.
 - The auto-increment values of the restored tables for a partial restore might not be the same as they were at the end of the backup process.
 - Encrypted InnoDB tables cannot be included in a partial restore.

The following command restores the table `cats` in the `pets` schema from the backup:

Example 5.7 Restoring A Selected Table from an Image Backup

```
mysqlbackup --socket=/tmp/restoreserver.sock --include-tables="^pets\.cats" --backup-dir=/dba/backuptmp
--backup-image=/dba/my.mbi copy-back-and-apply-log
```

The following command restores all tables in the “sales” database from the backup, but excludes the table with the name “hardware” :

Example 5.8 Restoring Selected Tables in a Schema from an Image Backup

```
mysqlbackup --socket=/tmp/restoreserver.sock --include-tables="^sales\." \
--exclude-tables="^sales\.hardware$" --backup-dir=/dba/backuptmp --backup-image=/dba/my.mbi \
copy-back-and-apply-log
```

Also see [Section 5.1.5, “Restoring Backups Created with the `--use-tts` Option”](#) for additional information on partial restores using TTS backups.



Note

For MySQL Enterprise Backup 8.0.19 and before, partial restore is only supported for [TTS backups](#).

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

Requirements for restoring backups created with [transportable tablespaces \(TTS\)](#) (that is, created with the `--use-tts` option) are similar to those listed in [Section 5.1.4, “Table-Level Recovery \(TLR\)”](#), with some differences noted in the section. The following is some additional information on partial restores using TTS backups

To backup and restore backups created using TTS, 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.

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.

You can rename a table when restoring it from a TTS backup by using the `--rename` option (the option is not supported for [non-TTS backups](#)):

Example 5.9 Restoring and Renaming a Table from a TTS Backup

```
# Using fully qualified table names:
mysqlbackup --socket=/tmp/restoreserver.sock \
--backup-dir=/BackupDirTemp --backup-image=/home/dbadmin/backups/tts-backup.mbi \
--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 \
--backup-dir=/BackupDirTemp --backup-image=/home/dbadmin/backups/tts-backup.mbi \
--include-tables="^sales\.cars" --rename="cars to autos" copy-back-and-apply-log
```

5.1.6 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 5.10 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 5.11 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 5.12 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 5.13 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 5.14 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
```

5.1.7 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 `tablespace_tracker` file inside the backup; see description of the file in Table 1.1, “Types of Files in a Backup” for details.

5.1.8 Advanced: Preparing and Restoring a Directory Backup

A directory backup, just like a single-file backup, can be [prepared](#) and restored using the `copy-back-and-apply-log` command as explained at the beginning of Section 5.1, “Performing a Restore Operation”.

Example 5.15 Restoring a Backup Directory using `copy-back-and-apply-log`

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

However, two alternatives exist for directory backups:

- Perform the [apply log operation](#) on the [raw backup](#) right after the backup, or anytime before restore, using the `apply-log` command. 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. Here are some examples of doing that, on different kinds of directory backups:

Example 5.16 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 5.17 Applying the Log to a Compressed Backup

If the backup is compressed, as in Section 4.3.4, “Making a Compressed Backup”, specify the `--uncompress` option to `mysqlbackup` when applying the log to the backup (the `--uncompress` option is required only for MySQL Enterprise Backup 8.0.20 and earlier):

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



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 backups that are non-incremental, you can combine the initial backup and the `apply-log` steps using the `backup-and-apply-log` command.

After the backup has been prepared, you can now restore it using the `copy-back` command:

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

5.2 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 (which is true by default for MySQL 8.0). 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- $t_R$ " \
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-to- $t_R$ " \
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](#).

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

5.3 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 8.0.11 to 5.7.21) might cause server crashes or data corruption.



Important

Currently, server downgrade, even within the same release series, is not supported, as per the policy described in [Downgrading MySQL](#)

- *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.



Note

For MySQL 8.0.15 and earlier: Make sure the `mysql_upgrade` that comes with your newer server version is applied (there is no need to run `mysql_upgrade` after upgrading to MySQL 8.0.16 or later).

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

Chapter 6 Working with Encrypted InnoDB Tablespaces

MySQL Enterprise Backup supports encrypted InnoDB tablespaces. For details on how the MySQL server encrypts and decrypts InnoDB tablespaces, see [InnoDB Data-at-Rest Encryption](#)—it explains concepts like master key and tablespace keys, which are important for understanding how MySQL Enterprise Backup works with encrypted InnoDB tablespaces.

When InnoDB tablespace encryption uses Oracle Key Vault (OKV) for encryption key management, the feature is referred to as “MySQL Enterprise Transparent Data Encryption (TDE).”

The following is a brief description on how encrypted InnoDB tablespaces are handled by MySQL Enterprise Backup in backup, restore, and apply-log operations.



Note

- For MySQL Enterprise Backup 8.0.16 and later: [Encrypted InnoDB undo logs](#) are supported by MySQL Enterprise Backup. The encrypted undo tablespaces are handled the same way as the encrypted tablespaces for InnoDB tables.
- For MySQL Enterprise Backup 8.0.17 and later: [Encrypted InnoDB redo logs](#) are supported by MySQL Enterprise Backup. The encrypted redo tablespaces are handled the same way as the encrypted tablespaces for InnoDB tables.

Backing up a database with encrypted InnoDB tablespaces.



Important

For MySQL Enterprise Backup to backup encrypted InnoDB tablespaces, the operating system user that runs MySQL Enterprise Backup must have write permission for the keyring file on the server if the [keyring_file](#) or [keyring_aws](#) plugin is used on it.

When the database uses encrypted InnoDB tablespaces, MySQL Enterprise Backup always stores the master key for encryption in an encrypted file inside the backup, irrespective of the kind of keyring plugin the server uses. The following is a typical command for backing up a database containing encrypted InnoDB tablespaces:

```
$ mysqlbackup --user=root --password --backup-image=/home/admin/backups/my.mbi --backup-dir=/home/admin/backups --encrypt-password="password" backup-to-image
```

During the backup operation, [mysqlbackup](#) copies the encrypted InnoDB tablespace files into the backup, and also performs the following actions:

- [mysqlbackup](#) contacts the MySQL server to determine the keyring plugin the server is using, which, currently, is either one of [keyring_encrypted_file](#), [keyring_file](#), [keyring_okv](#), or [keyring_aws](#).
- If the server is using the [keyring_encrypted_file](#) plugin, the user must use the option [--encrypt-password](#) to supply to [mysqlbackup](#) the keyring file encryption password that has been set on the server with the [--keyring_encrypted_file_password](#) option. [mysqlbackup](#) then copies over from the server the encrypted keyring data file, which contains the master key used to encrypt all the tablespace keys, into the [meta](#) folder in the backup. The encrypted tablespace files are also copied into the backup.
- If the server uses a keyring plugin other than [keyring_encrypted_file](#), [mysqlbackup](#) accesses the keyring to obtain the master key and uses it to decrypt the encrypted tablespace keys, which were used to encrypt the InnoDB tablespaces on the server. The master key is then put into a keyring data file, named [keyring_kef](#) and saved in the [meta](#) folder in the backup; the file is encrypted with the user password supplied with the option [--encrypt-password](#).



Note

Users who do not want to supply the password on the command line or in a defaults file may use the `--encrypt-password` option without specifying any value; `mysqlbackup` then asks the user to type in the password before the operation starts. This applies to all commands that use the `--encrypt-password` option.

An `extract` or `image-to-backup-dir` command for an image backup containing encrypted InnoDB tablespaces does not require the `--encrypt-password` option.

Restoring a backup with encrypted InnoDB tablespaces. The following is a typical command for restoring a single-file back up containing encrypted InnoDB tablespaces:

```
$ mysqlbackup --defaults-file=/usr/local/mysql/my.cnf --backup-image=/home/admin/backups/my.mbi \
--backup-dir=/home/admin/restore-tmp --encrypt-password="password" copy-back-and-apply-log
```

The same password used for backing up the database must be supplied with the `--encrypt-password` option for a restore operation. During a restore, `mysqlbackup` copies the encrypted InnoDB tablespace files and the encrypted file containing the master keys (`keyring_kef`) onto the server. It also performs the following actions:

- For a MySQL Enterprise Server: `mysqlbackup` restores the encrypted keyring data file to its proper location on the server. The restored server has to be started with `keyring_encrypted_file` plugin and with the options `keyring_encrypted_file_data` and `--keyring_encrypted_file_password` (which should supply the server with the same password used with the `--encrypt-password` option during the restore).
- For a MySQL Community Server: The `keyring_file` plugin is the only keyring plugin supported by the MySQL Community Server; therefore `mysqlbackup` uses the password supplied with the `--encrypt-password` option to decrypt keyring data file and then restores it to the proper location on the server for the `keyring_file` plugin to use.

For Incremental Backups. For a series of incremental backups, if a keyring plugin other than `keyring_encrypted_file` is being used on the server, users can provide a different value for `--encrypt-password` for any of the full or incremental backup in the backup sequence. However, the password used to make the specific full or incremental backup must be provided to restore that backup. When starting the server after restoring a series of incremental backups, the password used for the restore of the last incremental backup should be supplied to the server (except for a MySQL Community Server, which will start with the `keyring_file` plugin and does not require the `--keyring_encrypted_file_password` option to start).

Advanced: Creating and Restoring a directory backup with encrypted InnoDB tablespaces. The following is a typical command for creating a directory backup containing encrypted InnoDB tablespaces:

```
$ mysqlbackup --user=root --password --backup-dir=/home/admin/backup \
--encrypt-password="password" backup
```

The following is a typical command for preparing the backup with the `apply-log` command:

```
$ mysqlbackup --backup-dir=/home/admin/backup --encrypt-password="password" apply-log
```

Notice that the user password must be supplied with the `--encrypt-password` option, as the tablespace keys and then the tablespaces must be decrypted before the log can be applied. The same requirement applies when you try to update an encrypted backup with an encrypted incremental backup using the `apply-incremental-backup` command:

```
$ mysqlbackup --backup-dir=/home/admin/backup --incremental-backup-dir=/home/admin/backup-in \
--encrypt-password="password" apply-incremental-backup
```

If you used different values for `--encrypt-password` for the full or incremental backups in the backup sequence, make sure you supply the very password you used to create an individual backup when you perform an `apply-log` or `apply-incremental-backup` operation with it.

Next, a `copy-back` command restores the prepared backup onto the server:

```
$ mysqlbackup --defaults-file=/usr/local/mysql/my.cnf --backup-dir=/home/admin/backup copy-back
```

Notice that the `--encrypt-password` option is not required for this step.

You can combine the two steps of `apply-log` and `copy-back` into one by running the `copy-back-and-apply-log` command, for which the `--encrypt-password` option is required:

```
$ mysqlbackup --defaults-file=/usr/local/mysql/my.cnf --backup-dir=/home/admin/backup \
--encrypt-password="password" copy-back-and-apply-log
```

Limitations. Certain limitations apply when MySQL Enterprise Backup works with encrypted InnoDB tablespaces:

- *For MySQL Enterprise Backup 8.0.20 and earlier:* During a `validate` operation, if `mysqlbackup` encounters any encrypted InnoDB tablespaces, it issues a warning and then skips over them.
- *For MySQL Enterprise Backup 8.0.20 and earlier:* For partial backups using transportable table spaces (that is, when the `--use-tts` option is used), encrypted InnoDB tables are never included in a backup. A warning is issued in the log file whenever an encrypted InnoDB table that matches the table selection criteria has been skipped over.
- The `--skip-unused-pages` option has no effect on encrypted InnoDB tables during a backup (that is, empty pages for those tables are not skipped).
- If the server performs a `master key rotation` when a backup is running, the resulting backup might become corrupted.

Chapter 7 Backing up Using Redo Log Archiving

For MySQL Enterprise Backup 8.0.17 and later: `mysqlbackup` may sometimes fail to keep pace with redo log generation on the backed up server while a backup operation is in progress, resulting in lost redo log records due to those records being overwritten. This issue most often occurs when there is significant server activity during the backup operation, and the redo log file storage media operates at a faster speed than the backup storage media. The `redo log archiving` feature, introduced in MySQL 8.0.17, addresses this issue by sequentially writing redo log records to an archive file in addition to the redo log files. `mysqlbackup` can then copy redo log records from the archive file as necessary, thereby avoiding the potential loss of data.

When `redo log archiving` has been enabled on the server to be backed up, `mysqlbackup` utilizes the feature by default for backups, as long as the following are true:

- The OS user that runs `mysqlbackup` has read and write access to the folder on the server that stores the redo log archive file; the folder is the first labeled directory defined by the system variable `innodb_redo_log_archive_dirs` on the server.
- The MySQL user `mysqlbackup` uses to connect to the server must be granted the `INNODB_REDO_LOG_ARCHIVE` privilege, for activating `redo log archiving` on the server.

Redo log archiving can be skipped using the `mysqlbackup` option `--no-redo-log-archive`.

If `redo log archiving` is skipped by `mysqlbackup`, disabled on the server, or is simply not working for some reasons, `mysqlbackup` then reverts to copying the redo log data from the redo log files, as it used to do before the feature was introduced.

Chapter 8 Using MySQL Enterprise Backup with Replication

Table of Contents

8.1 Setting Up a New replica	81
8.2 Backing up and Restoring a Replica Database	83
8.3 Restoring a Source Database	83
8.4 Working with Encrypted Binary and Relay Logs	85

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.

8.1 Setting Up a New replica

MySQL Enterprise Backup allows you to set up a replica server (referred to as the “replica” below) by backing up the source server (referred to as the “source” below) and restoring the backup on a new replica, 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`. 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 (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=186
```

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](#).

8.2 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 8.1, “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 read lock on all the non-InnoDB tables and copy them.
 - 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.

8.3 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's `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](#) 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;
```

8.4 Working with Encrypted Binary and Relay Logs

MySQL Enterprise Backup 8.0.14 and later supports [encrypted binary and relay logs](#), which are handled in a similar way as the encrypted InnoDB tables are (see [Chapter 6, Working with Encrypted InnoDB Tablespace](#)s for details).

When backing up encrypted binary or relay logs, the option `--encrypt-password` is required for the following purposes:

- If the server is using the `keyring_encrypted_file` plugin, the user must use the option `--encrypt-password` to supply to `mysqlbackup` the keyring file encryption password that has been set on the server with the `keyring_encrypted_file_password` option. `mysqlbackup` then copies from the server the encrypted keyring data file, which contains the replication master key used to encrypt all the passwords for the individual log files, into the `meta` folder in the backup.
- If the server uses a keyring plugin other than `keyring_encrypted_file`, `mysqlbackup` accesses the keyring to obtain the replication master key and uses it to decrypt the individual log files' passwords. The replication master key is then put into a keyring data file, which is encrypted with the user password supplied with the option `--encrypt-password`, and then saved under the `meta` folder in the backup with the name `keyring_kef`.

When restoring encrypted binary or relay logs, the same password used for backing up the database must be supplied with the `--encrypt-password` option, as `mysqlbackup` performs the following actions:

- For a MySQL Enterprise Server: `mysqlbackup` restores the encrypted keyring data file to its proper location on the server. The restored server has to be started with `keyring_encrypted_file` plugin and with the options `keyring_encrypted_file_data` and `keyring_encrypted_file_password` (which should supply the server with the same password used with the `--encrypt-password` option during the restore).
- For a MySQL Community Server: The `keyring_file` plugin is the only keyring plugin supported by the MySQL Community Server; therefore `mysqlbackup` uses the password supplied with the `--encrypt-password` option to decrypt the keyring data file and then restores it to the proper location on the server for the `keyring_file` plugin to use.

For Incremental Backups. For a series of incremental backups, if a keyring plugin other than `keyring_encrypted_file` is being used on the server, users can provide a different value for `--encrypt-password` for any of the full or incremental backup in the backup sequence. However, the password used to make the specific full or incremental backup must be provided to restore that backup. When starting the server after restoring a series of incremental backups, the password used for the restore of the last incremental backup should be supplied to the server (except for a MySQL Community Server, which will start with the `keyring_file` plugin and does not require the `keyring_encrypted_file_password` option to start).

Chapter 9 Using MySQL Enterprise Backup with Group Replication

For how to use MySQL Enterprise Backup to back up and subsequently restore a [Group Replication](#) member, see [Using MySQL Enterprise Backup with Group Replication](#).

For MySQL Enterprise Backup 8.0.12 and later: When working with a [Group Replication](#) setup, `mysqlbackup` makes the backup history available to all members of the server group by making sure that the `backup_history` table is updated on a primary node after each `mysqlbackup` operation. The feature requires the following to be true:

- Host names or host addresses contained in the `member_host` column in the `performance_schema.replication_group_members` table must be resolvable by `mysqlbackup`.
- The same user has been created on every member of the server group to allow `mysqlbackup` to access the server from any host in the group. This is a sample statement for creating such a user on every member host:

```
CREATE USER 'mysqlbackup'@'host_name' IDENTIFIED BY 'password';
```

Do not use `localhost` for `host_name`, as that prevents `mysqlbackup` from connecting from another member host of the group.

- All the required privileges, as described in [Section 4.1.2, “Grant MySQL Privileges to Backup Administrator”](#), have been granted to the user with which `mysqlbackup` connects to the servers, and in the following manners:
 - Privileges are granted to the user created as discussed in the last bullet (`'mysqlbackup'@'host_name'` in the example), not to `'mysqlbackup'@'localhost'` as illustrated in [Section 4.1.2, “Grant MySQL Privileges to Backup Administrator”](#).
 - For any of the privileges, make sure it is granted for the user to connect from any member host. For example, on each host, the grant statement for `SELECT` on `performance_schema.replication_group_members` should be repeated to allow access from all member hosts of the group (which are `host1`, `host2`, and `host3` in this example):

```
GRANT SELECT ON performance_schema.replication_group_members TO 'mysqlbackup'@'host1';
GRANT SELECT ON performance_schema.replication_group_members TO 'mysqlbackup'@'host2';
GRANT SELECT ON performance_schema.replication_group_members TO 'mysqlbackup'@'host3';
```

Or, if it is possible and secure for your setup, use a wildcard to do the same with one statement; for example:

```
GRANT SELECT ON performance_schema.replication_group_members TO 'mysqlbackup'@'host_%';
```

Chapter 10 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
6ald325e6ef0577f3400b7cd624ae574f5186d0da2eeb946895de418297ed75b
```

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 MBackupP\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 19.13, “Encryption Options”](#) for details.

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

```
mysqlbackup --backup-image=/backups/image.enc --encrypt
--key=23D987F3A047B475C900127148F9E0394857983645192874A2B3049570C12A34
--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 11 Using MySQL Enterprise Backup with Media Management Software (MMS) Products

Table of Contents

11.1 Backing Up to Tape with Oracle Secure Backup	91
---------------------------------------------------------	----

This section describes how you can use MySQL Enterprise Backup in combination with media management software (MMS) products for creating backups for your database. Such products are typically used for managing large volumes of backup data, often with high-capacity backup devices such as tape drives. Also see [this whitepaper](#) for more discussions on using an MMS product with `mysqlbackup` through the System Backup to Tape (SBT) interface.

11.1 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) API.

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 19.9, “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.

Each time an online backup is made to a tape using the SBT API, besides recording the backup in the `mysql.backup_history` and the `mysql.backup_progress` tables, an entry is also made

to the `mysql.backup_sbt_history` table on the backed up MySQL instance. That facilitates the management of tape backups by allowing easy look-ups for information on them. The definition of the `backup_sbt_history` table is shown below:

```
mysql> DESCRIBE `backup_sbt_history`;
```

Field	Type	Null	Key	Default	Extra
id	int	NO	PRI	NULL	auto_increment
backup_id	bigint	NO		NULL	
backup_file_name	varchar(4096)	NO		NULL	
file_creation_time	timestamp	NO		0000-00-00 00:00:00	
file_expiry_time	timestamp	NO		0000-00-00 00:00:00	
volume_label	varchar(64)	NO		NULL	
sbt_error_msg	varchar(4096)	NO		NULL	
sbt_error_code	int	NO		NULL	

Here are the descriptions for the fields in the table:

- `id`: Auto-increment primary key for the table.
- `backup_id`: The backup's ID, which is also recorded in the backup's entries in the `mysql.backup_history` and the `mysql.backup_progress` tables.
- `backup_file_name`: The file name provided by the user through the `--backup-image=sbt:name` option.
- `file_creation_time`: Creation date and time for the tape backup.
- `file_expiry_time`: Expiration date and time for the tape backup.
- `volume_label`: Volume label for the physical medium which contains the tape backup.
- `sbt_error_msg`: Error message, when an error occurs while retrieving information for the tape backup.
- `sbt_error_code`: Error code, when an error occurs while retrieving information for the tape backup.

Multiple entries, one for each volume label, are created in the `mysql.backup_sbt_history` table, if the backup is split across multiple volumes.

Here are some sample entries in the `mysql.backup_sbt_history` table:

```
mysql> SELECT * FROM mysql.backup_sbt_history;
```

id	backup_id	backup_file_name	file_creation_time	file_expiry_time	volume_label
1	15921945689894983	backup_img1.msb	2020-06-15 07:16:09	2020-06-15 07:16:09	/sbt_bup_dir
2	15921945689894983	backup_img1.msb	2020-06-15 07:16:09	2020-06-15 07:16:09	backup_img1.msb

2 rows in set (0.00 sec)

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 which was used during the backup. This single parameter retrieves the appropriate data from the appropriate tape device.
- Optionally use the `--sbt-lib-path` option, using the same value 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 11.1 Sample `mysqlbackup` Commands Using MySQL Enterprise Backup with Oracle Secure Backup

```
# Uses libobk.so or ORASBT.DLL, at standard locations:
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 12 Using MySQL Enterprise Backup with Docker

MySQL Enterprise Backup is included in the Docker installation of MySQL Enterprise Edition. See [Using MySQL Enterprise Backup with Docker](#) in the [MySQL 8.0 Reference Manual](#) for usage instructions.

Chapter 13 Performance Considerations for MySQL Enterprise Backup

Table of Contents

13.1 Optimizing Backup Performance	97
13.2 Optimizing Restore Performance	100

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

13.1 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 13.2, “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 19.7, “Incremental Backup Options”](#) for information about these options. For usage instructions for the backup and apply stages of incremental backups, see [Section 4.3.3, “Making a Differential or Incremental 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 4.3.4, “Making a Compressed Backup”](#) and [Section 19.6, “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.

InnoDB Configuration Options Settings

It used to be a 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. Since MySQL 5.5, the performance of [crash](#)

`recovery` has been improved significantly, as explained in [Optimizing InnoDB Configuration Variables](#). 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 19.10, “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 `iowait` 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 `.sdi` files) temporarily puts those tables into a read-only state, using the statement `FLUSH TABLES tbl_name [, tbl_name] ... WITH READ LOCK`. For best backup performance and minimal impact on database processing:

1. Do not run long `INSERT`, `UPDATE`, or `DELETE` queries 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 time where non-InnoDB tables are read-locked will be short and the normal processing of `mysqld` will not be disturbed 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. Note

that files copied under the `--no-locking` setting cannot be guaranteed to have consistent data.

- 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!”.
- 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 19.8, “Partial Backup and Restore Options”](#) for information about the relevant options, and [Section 4.3.5, “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 4.3.5, “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.

Advanced: Apply-Log Phase (for Directory Backups only)

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. [Apply-log Operation](#)

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 [Apply-log Operation](#).

13.2 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 18.3, “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 4.3.5, “Making a Partial Backup”](#) for information about 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 19.6, “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 18.5, “Other Single-File Backup Operations”](#) for information about making single-file backups.

The Apply-Log Phase (for Directory Backups only)

See [Advanced: Apply-Log Phase \(for Directory Backups only\)](#) 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 19.10, “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.

For an apply-log operation, the `--process-threads` option controls the number of threads that read and write modified datafile pages in parallel; those threads are usually I/O bound, even though they also perform some in-memory processing.

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 14 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 15 Using MySQL Enterprise Backup with MySQL Enterprise Firewall

MySQL Enterprise Edition includes MySQL Enterprise Firewall, an application-level firewall that enables database administrators to permit or deny SQL statement execution based on matching against allowlists of accepted statement patterns.

When using MySQL Enterprise Firewall with MySQL Enterprise Backup, record the statement `FLUSH ENGINE LOGS` explicitly in the statement allowlist; otherwise some MySQL Enterprise Backup might then fail. Because MySQL Enterprise Backup 8.0 only uses the statement sporadically, it is easy to miss it in the training phase for the MySQL Enterprise Firewall.

Chapter 16 Troubleshooting for MySQL Enterprise Backup

Table of Contents

16.1 Exit codes of MySQL Enterprise Backup	107
16.2 Working Around Corruption Problems	108
16.3 Using the MySQL Enterprise Backup Logs	109
16.4 Using the MySQL Enterprise Backup Manifest	110

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's documentation for the meaning of those error codes and how to handle them.
- 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 19.11, "Message Logging Options"](#) for details on how to configure this logging feature.
- Incremental backups, when performed using the `--start-lsn` option, 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.

16.1 Exit codes of MySQL Enterprise Backup

MySQL Enterprise Backup returns one of the following exit codes as it exists an operation. The meaning of each code is explained in [Table 16.1, "MySQL Enterprise Backup Exit Codes and Messages"](#) by its associated exit message.

Table 16.1 MySQL Enterprise Backup Exit Codes and Messages

Exit Code	Exit Message
0	No error
1	Unknown Error
2	Internal Error
3	One of the required files is corrupt
4	One of the required files not found
5	Corrupt page or header encountered
6	Mismatch in config and the value obtained
7	Illegal Argument
8	One or more of the arguments are unknown
9	IO operation failed

Exit Code	Exit Message
10	Error allocating memory
11	Connection to server failed
12	Ongoing operation interrupted by user
13	User doesn't have sufficient privileges
14	No space left on device
15	Image version is not supported by this version of meb
16	The value is out of range
17	Innodb Error
18	Timedout while waiting for resource
19	Server returned error while executing sql

The `mysqlbackup` command `print-message` takes an exit code supplied with the `--error-code` option and returns the corresponding exit message in the `stdout` stream. Users can, for example, use a script to catch the exit code returned by `mysqlbackup`, and then pass it onto the `print-message` command to obtain an exit message. See the description for `print-message` for details.

16.2 Working Around Corruption Problems

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

```
170225 10:46:18 PCR1 INFO: Re-reading page at offset 0 in D:/temp/5.7_source/test/emp2.ibd
170225 10:46:18 PCR1 INFO: Re-reading page at offset 0 in D:/temp/5.7_source/test/emp2.ibd
...
170225 10:46:26 PCR1 ERROR: Page at offset 0 in D:/temp/5.7_source/test/emp2.ibd 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.

16.3 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
id	int	NO	PRI	NULL	auto_increment
backup_id	bigint	NO	MUL	NULL	
tool_name	varchar(4096)	NO		NULL	
error_code	int	NO		NULL	
error_message	varchar(4096)	NO		NULL	
current_time	timestamp(3)	NO		CURRENT_TIMESTAMP(3)	DEFAULT_GENERATED on update CURRENT_TIMESTAMP(3)
current_state	varchar(200)	NO		NULL	

7 rows in set (0.00 sec)

For MySQL Enterprise Backup 8.0.19 and later: The `backup_progress` table is in InnoDB format.

For MySQL Enterprise Backup 8.0.18 and earlier: 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).
- Check the `tool_name` column for the full `mysqlbackup` command that triggered the backup, including all the options used.
- 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> mysql> DESCRIBE mysql.backup_history;
```

Field	Type	Null	Key	Default	Extra
backup_id	bigint(20)	NO	PRI	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		
start_lsn	bigint(20)	NO		NULL		
end_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		
server_uuid	varchar(36)	NO		NULL		

+-----+-----+-----+-----+-----+-----+-----+

28 rows in set (0.01 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 triggered 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`, `DIFFERENTIAL`, `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.

16.4 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 backup. 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

17	mysqlbackup	117
18	mysqlbackup commands	119
	18.1 Backup Operations	119
	18.2 Update Operations	120
	18.3 Restore Operations	121
	18.4 Validation Operations	124
	18.5 Other Single-File Backup Operations	124
	18.6 Other Operations	126
19	mysqlbackup Command-Line Options	129
	19.1 Standard Options	134
	19.2 Connection Options	136
	19.3 Server Repository Options	137
	19.4 Backup Repository Options	139
	19.5 Metadata Options	142
	19.6 Compression Options	143
	19.7 Incremental Backup Options	145
	19.8 Partial Backup and Restore Options	148
	19.9 Single-File Backup Options	152
	19.10 Performance / Scalability / Capacity Options	155
	19.11 Message Logging Options	161
	19.12 Progress Report Options	163
	19.13 Encryption Options	166
	19.14 Options for Working with Encrypted InnoDB Tablespaces and Encrypted Binary/Relay Logs	166
	19.15 Cloud Storage Options	167
	19.16 Options for Special Backup Types	170
	19.17 Other Options	173
20	Configuration Files and Parameters	175

Chapter 17 `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 `.sdi` files that record the structure of MyISAM tables.
- 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 19, `mysqlbackup` Command-Line Options](#). For information about configuration files, see [Chapter 20, `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.

Review [Section 13.1, “Optimizing Backup Performance”](#) to understand the various issues that can impact the performance of MySQL Enterprise Backup.

Chapter 18 `mysqlbackup` commands

Table of Contents

18.1 Backup Operations	119
18.2 Update Operations	120
18.3 Restore Operations	121
18.4 Validation Operations	124
18.5 Other Single-File Backup Operations	124
18.6 Other Operations	126

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`
- Update 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`

18.1 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]
            [SINGLE-FILE-BACKUP-OPTIONS]
            [PERFORMANCE-SCALABILITY-CAPACITY-OPTIONS]
            [MESSAGE-LOGGING-OPTIONS]
            [PROGRESS-REPORT-OPTIONS]
            [ENCRYPTION-OPTIONS]
            [CLOUD-STORAGE-OPTIONS]
            [ENCRYPTED-INNOODB-OPTIONS]
            backup-to-image
```

```
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]
            [ENCRYPTED-INNODB-OPTIONS]
            backup | backup-and-apply-log
```

- [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.

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 location specified by [--backup-dir](#); if the [--with-timestamp](#) option is also used, the backup image is then saved in a subdirectory created under the folder specified by [--backup-dir](#) that bears the timestamp in its name.

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.

- [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.

18.2 Update Operations

There are two types of operations to bring your backup data up-to-date:

Apply-log Operation

After a backup job was first completed, the backup data might not be in a consistent state, because data could have been inserted, updated, or deleted while the backup was running. This initial backup file is known as the [raw backup](#). During a backup, [mysqlbackup](#) also copies the accumulated InnoDB log to a file called [ibbackup_logfile](#). In an apply-log operation, the [ibbackup_logfile](#) file is used to “roll forward” the raw data files, so that every page in the data files corresponds to the same

log sequence number of the InnoDB log. This is similar to the operation that takes place during a [crash recovery](#).

For single-file backups, the apply-log operation is usually performed as part of the [copy-back-and-apply-log](#) command. For directory backups, the [copy-back-and-apply-log](#) command can also be used, but you also have the two alternatives of

- Performing the apply-log operation together with the back up using the [backup-and-apply-log](#) command (not applicable for incremental or compressed directory backups)
- Performing the apply-log operation separately with the [apply-log](#) command on the raw backup, before running the [copy-back](#) command.

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

- [apply-log](#)

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

Example 18.1 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.

Apply-incremental-backup Operation

Advanced: Use the [apply-incremental-backup](#) to update a backup directory with data in an incremental backup directory:

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

- [apply-incremental-backup](#)

Advanced: Brings up-to-date a directory backup specified with the [--backup-dir](#) option, using the data from an incremental backup directory specified with the [--incremental-backup-dir](#) option. See [Section 5.1.3, “Restoring an Incremental Backup”](#) for instructions on restoring incremental backups.

For a single-file incremental backup, you typically use the [copy-back-and-apply-log](#) command (with the [--incremental](#) option for MySQL Enterprise Backup 8.0.20 and earlier) to apply the data in the incremental image backup to the full backup that has already been restored to the data directory of the target server.

18.3 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. 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 a [partial restore](#). The option [datadir](#) must be specified either in the file specified by

the `--defaults-file` option or as a command-line option. For usage examples, see [Chapter 5, Recovering or Restoring a Database](#).

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

- `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`:

- *For MySQL Enterprise Backup 8.0.20 and earlier:* To restore a compressed directory or image, include the `--uncompress` option in the command line (the option is no longer required for MySQL Enterprise Backup 8.0.21 and later).
- 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 a single-file incremental backup, assuming the full backup (on which the incremental backup was based) has already been restored:
 - *For MySQL Enterprise Backup 8.0.20 and earlier:* Include the `--incremental` option in the command line (the option is no longer required for MySQL Enterprise Backup 8.0.21 and later).
 - *For MySQL Enterprise Backup 8.0.20 and earlier:* Include the `--uncompress` option in the command line if the incremental backup has been created with the `--compress` option (the option is no longer required for MySQL Enterprise Backup 8.0.21 and later).
- Specifies the location of the incremental backup image with the `--backup-image` option.
- Supplies with the `--backup-dir` option the location of a folder that will be used for storing temporary files produced during the restoration process.
- *Advanced:* To restore an incremental backup directory, assuming the full backup (on which the incremental backup was based) has already been restored:

- For MySQL Enterprise Backup 8.0.20 and earlier: Include the `--incremental` option in the command line (the option is no longer required for MySQL Enterprise Backup 8.0.21 and later).
- Use either the `--backup-dir` or `--incremental-backup-dir` option to specify the incremental backup directory.
- To restore selected tables:
 - See the general requirements described in [Section 5.1.4, “Table-Level Recovery \(TLR\)”](#).
 - 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.
 - *Advanced:* 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`.

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 5.1, “Performing a Restore Operation”](#).

- `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 5.1.8, “Advanced: Preparing and Restoring a Directory Backup”](#) 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 the discussions [here](#) for details.

There are some special requirements when restoring selected tables from backups; see [Section 5.1.4, “Table-Level Recovery \(TLR\)”](#) for details.



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.

18.4 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:

- If any `.ibd` files or `.sdi` files are missing from the data directory during a backup or have been deleted from a backup after the backup was made, the `validate` operation will not be able to detect the problem.
- If a backup 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.
- 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, `.sdi` file corruptions) are not detected.
- *For MySQL Enterprise Backup 8.0.20 and earlier only:* During a `validate` operation, if `mysqlbackup` encounters any encrypted InnoDB tablespaces, it issues a warning and then skips over them.

Here is a sample command for validating a backup image:

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

Advanced: Here is a sample command for validating a backup directory:

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

For more usage examples for the `validate` command, see [Section 4.2.2, “Verifying a Backup”](#)

18.5 Other Single-File Backup Operations

Besides the commands for creating and restoring single-file backups (namely, `backup-to-image` and `copy-back-and-apply-log`), `mysqlbackup` provides a number of other commands for you to work with single-file backups. They are explained below.

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

```

- [image-to-backup-dir](#)

For MySQL Enterprise Backup 8.0.18 and later: It is an alias for the [extract](#) command; see the description below for [extract](#).

For MySQL Enterprise Backup 8.0.17 and earlier: 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 4.3.1, “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 4.3.1, “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 4.3.1, “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 4.3.1, “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.



Tip

If you want to extract only from specific directories (for example, `datadir/meta`), add a slash at the end of the option value (`--src-entry=meta/`); otherwise, any file or directory in the backup that contains the value in its pathname (including, for example, `datadir/pets/metabolism.ibd`) will also be extracted.

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.

For MySQL Enterprise Backup 8.0.18 and later: Use the `--uncompress` option to extract files from a compressed single-file backup (the `--uncompress` option is not required for MySQL Enterprise Backup 8.0.21 and later *when the `--src-entry` is used*).

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 4.12, “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.

18.6 Other Operations

This group of operations consists of any `mysqlbackup` commands not covered in other sections of this chapter.

```
mysqlbackup --error-code=CODE print-message
```

- `print-message`

Prints the associated exit message for a `mysqlbackup` exit code to the `stdout` stream.

Use the `--error-code` option to supply the exit code for which you want to receive the associated exit message:


```
$ mysqlbackup print-message --error-code=4 2> /dev/null
```

```
One of the required files not found
```

For a list of `mysqlbackup` exit codes and messages, see [Section 16.1, “Exit codes of MySQL Enterprise Backup”](#).

Chapter 19 `mysqlbackup` Command-Line Options

Table of Contents

19.1 Standard Options	134
19.2 Connection Options	136
19.3 Server Repository Options	137
19.4 Backup Repository Options	139
19.5 Metadata Options	142
19.6 Compression Options	143
19.7 Incremental Backup Options	145
19.8 Partial Backup and Restore Options	148
19.9 Single-File Backup Options	152
19.10 Performance / Scalability / Capacity Options	155
19.11 Message Logging Options	161
19.12 Progress Report Options	163
19.13 Encryption Options	166
19.14 Options for Working with Encrypted InnoDB Tablespaces and Encrypted Binary/Relay Logs ..	166
19.15 Cloud Storage Options	167
19.16 Options for Special Backup Types	170
19.17 Other Options	173

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 20, 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 19.1 List of All Options

Option Name	Description	Introduced	Deprecated	Removed
--backup-dir	The directory to store the backup data.			
--backup-image	Specifies the path name of the backup image.			
--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.			
--backup_innodb_log_group_home_dir	Backup directory for InnoDB log files.			
--backup_innodb_undo_directory	The relative or absolute directory path where InnoDB creates separate tablespaces for the undo logs.			
--character-sets-dir	Directory for character set files.			

Option Name	Description	Introduced	Deprecated	Removed
<code>--cloud-access-key-id</code>	AWS access key ID for logging onto Amazon S3.			
<code>--cloud-aws-region</code>	Region for Amazon Web Services that mysqlbackup access for S3.			
<code>--cloud-basicauth-url</code>	The URL for HTTP Basic Authentication for accessing Swift.			
<code>--cloud-bucket</code>	The storage bucket on Amazon S3 for the backup image.			
<code>--cloud-buffer-size</code>	Size of buffer for cloud operations.			
<code>--cloud-ca-info</code>	Absolute path to the CA bundle file for host authentication for SSL connections.			
<code>--cloud-ca-path</code>	CA certificate directory, in addition to the system's default folder.			
<code>--cloud-chunked-transfer</code>	Use chunked transfer with cloud Swift service.			
<code>--cloud-container</code>	The Swift container for the backup image.			
<code>--cloud-identity-url</code>	The URL of the Keystone identity service.			
<code>--cloud-oauth-token</code>	The OAuth token for accessing Oracle Cloud Infrastructure Object Storage.	8.0.12		
<code>--cloud-object</code>	The Swift object for the backup image.			
<code>--cloud-object-key</code>	The Amazon S3 object key for the backup image.			
<code>--cloud-password</code>	Password for user specified by <code>--cloud-user-id</code> .			
<code>--cloud-proxy</code>	Proxy address and port number for overriding the environment's default proxy settings for accessing cloud service.			
<code>--cloud-region</code>	The Keystone region for the user specified by <code>--cloud-user-id</code> .			
<code>--cloud-secret-access-key</code>	AWS secret access key.			
<code>--cloud-service</code>	Cloud service for data backup or restoration.			
<code>--cloud-storage-url</code>	The Oracle Cloud Infrastructure Object Storage URL when using OAuth for client authentication.	8.0.12		
<code>--cloud-tempauth-url</code>	The URL of the identity service for authenticating user credentials with Swift's TempAuth authentication system.			
<code>--cloud-tenant</code>	The Keystone tenant for the user specified by <code>--cloud-user-id</code> .			
<code>--cloud-trace</code>	Print trace information for cloud operations.			
<code>--cloud-user-id</code>	User ID for accessing Swift.			
<code>--comments</code>	Specifies comments string.			
<code>--comments-file</code>	Specifies path to comments file.			
<code>--compress</code>	Create backup in compressed format.			
<code>--compress-level</code>	Specifies the level of compression.			

Option Name	Description	Introduced	Deprecated	Removed
--compress-method	Specifies the compression algorithm.			
--compression-algorithms	Permitted compression algorithms for connections to server	8.0.18		
--connect_timeout	Connection timeout in seconds.			
--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.			
--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.			
--encrypt-password	The user-supplied password by which mysqlbackup encrypts the encryption keys for encrypted InnoDB tablespaces.			
--error-code	The exit code for which the print-message command prints the corresponding exit message.			
--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			
--help	Display help.			
--host	Host name to connect.			
--include	[Legacy] Backup only those per-table innodb data files which match the regular expression REGEXP.		8.0.20	
--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.			

Option Name	Description	Introduced	Deprecated	Removed
--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_data_home_dir	Specifies base directory for all InnoDB data files in the shared system tablespace.			
--innodb_log_group_home_dir	The directory path to InnoDB log files.			
--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.			8.0.16
--log-bin	Specify the location for the binary log to be restored.			
--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.			
--messages-logdir	Specifies the path name of an existing directory for storing the message log.			
--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.			
--no-redo-log-archive	Skip using redo log archiving during backups.	8.0.17		
--number-of-buffers	Specifies the exact number of memory buffers to be used for the backup operation.			
--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-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.			

Option Name	Description	Introduced	Deprecated	Removed
<code>--optimistic-time</code>	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.			
<code>--page-reread-count</code>	Maximum number of page re-reads.			
<code>--page-reread-time</code>	Wait time before a page re-read.			
<code>--password</code>	Connection password.			
<code>--pipe</code>	alias for <code>--protocol=pipe</code> .			
<code>--port</code>	TCP portnumber to connect to.			
<code>--print-defaults</code>	Print a list of option values supplied by defaults files and exit.			
<code>--process-threads</code>	Specifies the number of process-threads for the backup operation.			
<code>--progress-interval</code>	Interval between progress reports in seconds.			
<code>--protocol</code>	Connection protocol.			
<code>--read-threads</code>	Specifies the number of read-threads for the backup operation.			
<code>--relay-log</code>	Specify the location for the relay log to be restored on a replica server.			
<code>--relay-log-index</code>	Specifies the absolute path of the index file that lists all the relay log files.			
<code>--rename</code>	Rename a single table when it is selected by the <code>--include-tables</code> option to be restored			
<code>--safe-slave-backup-timeout</code>	When backing up a replica server, the timeout value for waiting for the replication SQL thread to drop its temporary tables.			
<code>--sbt-database-name</code>	Used as a hint to the Media Management Software (MMS) for the selection of media and policies for tape backup.			
<code>--sbt-environment</code>	Comma separated list of environment variable assignments to be given to the SBT library.			
<code>--sbt-lib-path</code>	Path name of the SBT library used by software that manages tape backups.			
<code>--shared-memory-base-name</code>	It designates the shared-memory name used by a Windows server to permit clients to connect using shared memory (Windows only).			
<code>--show-progress</code>	Instructs mysqlbackup to periodically output short progress reports known as progress indicators on its operation.			
<code>--skip-binlog</code>	Do not include binary log files during backup, or do not restore binary log files during restore.			
<code>--skip-final-rescan</code>	Skip the final rescan for InnoDB tables that are modified by DDL operations.			

Option Name	Description	Introduced	Deprecated	Removed
<code>--skip-messages-logdir</code>	Disable logging to teelog file.			
<code>--skip-relaylog</code>	Do not include relay log files during backup, or do not restore relay log files during a restore.			
<code>--skip-unused-pages</code>	Skip unused pages in tablespaces when backing up InnoDB tables.			
<code>--slave-info</code>	Capture information needed to set up an identical replica server.			
<code>--sleep</code>	Time to sleep in milliseconds after copying each 1MB of data.			
<code>--socket</code>	Socket file to use to connect.			
<code>--src-entry</code>	Identifies a file or directory to extract from a single-file backup.			
<code>--ssl-ca</code>	CA file in PEM format (implies <code>--ssl</code>).			
<code>--ssl-capath</code>	CA directory (check OpenSSL docs, implies <code>--ssl</code>).			
<code>--ssl-cert</code>	X509 cert in PEM format (implies <code>--ssl</code>).			
<code>--ssl-cipher</code>	SSL cipher to use (implies <code>--ssl</code>).			
<code>--ssl-fips-mode</code>	Controls whether MEB operates in FIPS mode.	8.0.14		
<code>--ssl-key</code>	X509 key in PEM format (implies <code>--ssl</code>).			
<code>--ssl-mode</code>	Security state of connection to server.			
<code>--start-lsn</code>	Specifies the highest LSN value included in a previous backup.			
<code>--suspend-at-end</code>	Pauses the mysqlbackup command when the backup procedure is close to ending.			
<code>--trace</code>	Trace level of messages by mysqlbackup.			
<code>--uncompress</code>	Uncompress a backup during an operation.			
<code>--use-tts</code>	Enable selective backup of InnoDB tables using transportable tablespaces (TTS).			
<code>--user</code>	Database user name to connect.			
<code>--verbose</code>	Print more verbose information.			
<code>--version</code>	Display version information.			
<code>--with-timestamp</code>	Create a subdirectory underneath the backup directory with a name formed from the timestamp of the backup operation.			
<code>--write-threads</code>	Specifies the number of write-threads for the backup operation.			
<code>--zstd-compression-level</code>	Compression level for connections to server that use ZSTD compression	8.0.18		

19.1 Standard Options

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

- The following standard options also exist for the `mysql` command. Full descriptions for these options can be found in the MySQL reference manual, accessible through, e.g., [Server Option, 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 op
--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.

- Overwriting of InnoDB data and log files during the `apply-log` and `apply-incremental-backup` operations.
- Replacing of an image file during an `backup-to-image` or `backup-dir-to-image` operation.

`--trace=level`

Property	Value
Command-Line Format	<code>--trace=LEVEL</code>
Type	Enumeration
Default Value	0
Valid Values	0 1 2

Property	Value
	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)

19.2 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.



Notes

- `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.
- If none of the algorithms specified by `--compression-algorithms` are permitted by the server, connection to the server will not be established.

```
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
--password[=value] [ short option: -p ]
--connect-timeout
--ssl-mode=mode
--ssl-key=file_name
--ssl-cert=file_name
--ssl-ca=file_name
--ssl-capath=directory_name
```

```
--ssl-cipher=cipher_list
--ssl-fips-mode={OFF|ON|STRICT}
--tls-version= protocol_list
--compression-algorithms=name # For release 8.0.18 and later
--zstd-compression-level=number # For release 8.0.18 and later
```

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

19.3 Server Repository Options

These repository options specify various parameters related to the database server to which a backup is restored.

These options are used only with restore operations, that is, `copy-back` and `copy-back-and-apply-log`. The descriptions below explain how these options are used with `mysqlbackup`; for information about how these options are used with the MySQL server, click the option names to see the descriptions in the MySQL Reference Manual.

- `datadir=PATH`

This is the data directory for the restored MySQL server. It should be supplied with the `datadir` value of the target server for the restore.

This option must be specified for any restore operations, except for partial restores (see [Section 5.1.4, “Table-Level Recovery \(TLR\)”](#) for details).

- `log-bin[=basename]`

Specify the location for the binary log to be restored. By default, during a restore, the binary log is restored to the same location it was found on the backed-up server. Use this option to specify a different target location for the binary log. The option works similarly as the `--log-bin` option of the MySQL server in determining the location and the name of the binary log files—see [description of the server --log-bin option](#) for details. As a summary:

- Using this option without supplying a `basename` puts the binary log in the target server's data directory with the default basename `host_name-bin`.
- Using this option while supplying a `basename` puts the binary log in the target server's data directory with the specified basename.
- Using this option while supplying a `basename` containing a full file path (for example, `/home/admin/db/binlogdir/binlog`) puts the binary log on the target server in the specified folder (`/home/admin/db/binlogdir/`) using the supplied basename (`binlog`).

The option is only for the `copy-back-and-apply-log` and `copy-back` operations. Using it with any other operations makes the command fail.

- `relay-log[=basename]`

Specify the location for the relay log to be restored on a replica server. By default, during a restore for a replica server, the relay log is restored to the same location it was found on the backed-up replica server. Use this option to specify a different target location for the relay log. The option works similarly as the `--relay-log` option of the MySQL server in determining the location and the name of the relay log files—see [description of the server --relay-log option](#) for details. As a summary:

- Using this option without supplying a `basename` puts the relay log in the target server's data directory with the default basename `host_name-relay-bin`.
- Using this option while supplying a `basename` puts the relay log in the target server's data directory with the specified basename.

- Using this option while supplying a *basename* containing a full file path (for example, `/home/admin/db/relaylogdir/relaylog`) puts the relay log on the target server in the specified folder (`/home/admin/db/relaylogdir/`) using the supplied basename (`relaylog`).

The option is only for the `copy-back-and-apply-log` and `copy-back` operations. Using it with any other operations makes the command fail.

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

Specify the absolute path (including file name and extension) for restoring the index file that lists all the binary log files, if it is different from the default path given below.

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

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

Specify the absolute path (including file name and extension) for restoring the index file that lists all the relay log files, if it is different from the default path given below.

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

- `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.

For backups: You do not need to specify this option, because its value is retrieved automatically using the database connection.

For restores: The directory where InnoDB data files are to be restored. Specify the option only if the InnoDB data files are to be restored outside of the server's data directory. *For release 8.0.16 and later:* the specified directory must be non-existent or empty, or the restore operation will fail, even if the `--force` option is used.

- 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_log_group_home_dir=PATH`

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

For backups: You do not need to specify this option, because its value is retrieved automatically using the database connection.

For restores: The directory where InnoDB redo log files are to be restored. Specify the option only if the InnoDB redo log files are to be restored outside of the server's data directory. *For release 8.0.16 and later:* the specified directory must be non-existent or empty, or the restore operation will fail, even if the `--force` option is used.

- 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_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: You do not need to specify this option, because its value is retrieved automatically using the database connection.

For restores:

- *For release 8.0.16 and later:* The directory where the default InnoDB undo tablespaces, as well as any non-default undo tablespaces resided in the backed-up server's data directory, are to be restored. (External undo tablespaces are restored by default to the locations they were found on the backed-up server; see the description for [undo log files \[12\]](#) for details.) *The specified directory must be non-existent or empty, or the restore operation will fail, even if the `--force` option is used.*
- *For release 8.0.15 and earlier:* The directory where the 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.

19.4 Backup Repository Options

These options specify various parameters related to the backup image or directory, or to how the backup will be restored. Typically, `--backup-image` and `--backup-dir` are the only options 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-image=IMAGE`

Property	Value
Command-Line Format	<code>--backup-image=IMAGE</code>
Type	File name

Specify the path name of the file used for a single-file backup, restore, or another [single-file operation](#).

Except when streaming the backup image with `--backup-image=-`, if `--backup-image` does not give a full path name, this is how `mysqlbackup` interprets the value of the option:

- For `backup-to-image` operations, `mysqlbackup` takes the value of `--backup-image` as a path relative to the location specified by `--backup-dir`. If the `--with-timestamp` option is also used, the backup image is then saved in a subdirectory that bears the timestamp in its name under the [backup directory](#).
- For [single-file operations](#) other than `backup-to-image` *AND for release 8.0.19 and later*, `mysqlbackup` takes the value of `--backup-image` as a path relative to the current working directory in which the `mysqlbackup` command is run. If the `--with-timestamp` option is also used, the subdirectory that bears the timestamp in its name is created under the current directory.

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 4.3.1.2, “Backing Up to Tape”](#) for details about tape backups.

- `--backup_dir=PATH`

Same as `--backup-dir`. The [backup 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 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 single file whose name is specified with the `--backup-image` option.



Note

For release 8.0.18 and earlier and 8.0.21 and later: 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 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 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.

- *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.
- `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 the `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 `backup_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_dir` 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 by this option.

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; see the description for [undo log files \[12\]](#) on where undo logs are restored.

- `--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.

19.5 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 16.3, “Using the MySQL Enterprise Backup Logs”](#) for details about these tables.

Default: history logging is enabled.

- `--comments=STRING`

Property	Value
Command-Line Format	<code>--comments=STRING</code>
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`

Property	Value
Command-Line Format	<code>--comments-file=PATH</code>
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.

19.6 Compression Options

For an overview on backup compression, see [Section 4.3.4, “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.

You cannot use the `--compress` option together with the `--incremental-with-redo-log-only` option.

Default: compression is disabled.

- `--compress-method=ALGORITHM`

Property	Value
Command-Line Format	<code>--compress-method=ALGORITHM</code>
Type	Enumeration
Default Value	<code>lz4</code>
Valid Values	<code>zlib</code> <code>lz4</code> <code>lzma</code>

Property	Value
	<code>punch-hole</code>
	<code>none</code>

Specifies the algorithm for backup compression, or enables support for InnoDB [transparent page compression](#). The supported arguments for the option and the algorithms they represent are:

- **lz4**: LZ4 r109. Out of the three compression 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 compression 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 compression 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.
- **punch-hole**: (For MySQL Enterprise Backup 8.0.13 and later) Enables support for [transparent page compression](#) for InnoDB tables for directory backups, which means that when the target platform for the `mysqlbackup` backup or restore operation supports hole punching, `mysqlbackup` keeps the punched holes in the page-compressed InnoDB files it transfers.

Limitations: The feature is NOT supported in the following cases, for which punched holes are removed from the InnoDB files:

- For single-file backups.
- For [TTS](#), incremental, compressed, or encrypted backups.
- When a backup is not created in a file system (for example, when cloud storage is used to save the backup), or when the file system does not support sparse files.
- For those pages of InnoDB data files that are modified by the redo log in an [apply-log](#) operation.

When the feature is enabled but hole punching fails, `mysqlbackup` issues a warning message after the operation is completed; for example:

```
WARNING: "Punch hole" operation failed.
```

Or:

```
WARNING: InnoDB datafiles in the backup are larger than in the source because of missing sparse file s
```

A backup can be taken with `--compress-method=punch-hole` and then be restored later without using the feature; the reverse is also true: a backup taken without using `--compress-method=punch-hole` can be restored later with the feature.



Note

`punch-hole` is a special argument with the `--compress-method` option for supporting transparent page compression. `--compress-method=punch-hole` is ignored when used together with any other `mysqlbackup` compression options.

- `none`: No compression.

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

- `--compress-level=LEVEL`

Property	Value
Command-Line Format	<code>--compress-level=LEVEL</code>
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`

For MySQL Enterprise Backup 8.0.20 and earlier: When used with `copy-back`, `copy-back-and-apply-log`, `apply-log`, and `apply-incremental-backup`, performs uncompression on a compressed backup during the operation (the option is no longer needed for MySQL Enterprise Backup 8.0.21 and later).

For MySQL Enterprise Backup 8.0.18 and later: When used with the `extract` operation, uncompresses files that are extracted from a compressed single-file backup (the option is not required for MySQL Enterprise Backup 8.0.21 and later *when the `--src-entry` option is used*).

19.7 Incremental Backup Options

For an overview of incremental backups and usage examples for these options, see [Section 4.3.3, “Making a Differential or Incremental Backup”](#) and [Section 5.1.3, “Restoring an Incremental Backup”](#).

To take an incremental backup, specify the `--incremental` or `--incremental-with-redo-log-only`, along with the `--backup-dir` option. Depending on whether `--incremental` or `--incremental-with-redo-log-only` is used, other options are required or recommended. All InnoDB data modified after the a certain [LSN](#) (specified directly or indirectly by the options you use) is copied into the incremental backup. *For MySQL Enterprise Backup 8.0.20 and earlier:* to restore an incremental backup, specify the `--incremental` option (the option is no longer required for MySQL Enterprise Backup 8.0.21 and later for restore operations).

- `--incremental[={page-track|full-scan|optimistic}]`

Property	Value
Command-Line Format	<code>--incremental</code>
Type	Enumeration
Default Value	<code>full-scan</code>
Valid Values	<code>page-track</code> <code>full-scan</code>

Property	Value
	<code>optimistic</code>

When performing an incremental backup, there are three possible values for this option:

- **page-track:** For MySQL Enterprise Backup 8.0.18 and later: `mysqlbackup` looks for changed pages in the InnoDB data files that have been modified since the last backup using the page tracking functionality on the server and then copies them. This is potentially the fastest way for `mysqlbackup` to create incremental backups. Even with this value set, the page tracking functionality is only used when certain requirements are satisfied; see [Incremental Backup Using Page Tracking](#) for details.
- **full-scan:** `mysqlbackup` scans all InnoDB data files in the server's data directory to find pages that have been changed since the last backup and copies them.
- **optimistic:** `mysqlbackup` only scans for changed pages in the InnoDB data files that have been modified since the last backup and then copies them. In general, optimistic incremental backups are faster than full-scan ones when not many tables in the database have been modified; however a few restrictions apply to this feature. See [Full-scan versus Optimistic Incremental Backup](#) for details.

Default: `page-track`, for MySQL Enterprise Backup 8.0.18 and later. However, if the page tracking functionality cannot be utilized by `mysqlbackup` for some reasons (see [Incremental Backup Using Page Tracking](#) for details), `mysqlbackup` performs a full-scan backup instead if the `--incremental` option is not set, or throws an error when `--incremental=page-track`.

For MySQL Enterprise Backup 8.0.17 and earlier, full-scan backup is the default method for incremental backups, which is utilized if no value is specified for `--incremental`.

During a backup, the `--incremental` option also requires the use of either the `--incremental-base` option or the `--start-lsn` option. Only InnoDB tables are backed up incrementally. By default, all non-InnoDB 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.

The value for the option has only meaning when the option is used an incremental backup.

For MySQL Enterprise Backup 8.0.20 and earlier. For a `copy-back-and-apply-log`, `copy-back`, and `apply-log`, operation, specifies that the associated backup is **incremental** (the option is no longer required for MySQL Enterprise Backup 8.0.21 and later for restore operations).

- `--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`. Just like with the `--incremental` option, only InnoDB tables are backed up incrementally. By default, all non-InnoDB 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.

You cannot use the `--compress` option together with the `--incremental-with-redo-log-only` option.

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

Property	Value
Command-Line Format	<code>--incremental-base=mode:argument</code>
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 : argument` in the option syntax. The alternatives are:

- `history:[last_backup | last_full_backup]`

The prefix `history:` followed by one of the two possible values:

- `last_backup`: 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.
- `last_full_backup`: (For MySQL Enterprise Backup 8.0.17 and later) This works similarly as the value `last_backup`, except that it makes `mysqlbackup` look for the last `full backup` that was taken and use it as the base backup, thus creating a `differential backup`.



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.

- `dir:directory_path`

Advanced: You specify the prefix `dir:` followed by a directory path argument, which points to the previous directory backup. With the first incremental backup, you specify the directory holding the full directory backup; with the second incremental backup, you specify the directory holding the first incremental directory backup, and so on.

- `--start-lsn=LSN`

Property	Value
Command-Line Format	<code>--start-lsn=LSN</code>
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`

Advanced: 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.

19.8 Partial Backup and Restore 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 restore, as well as usage examples on the following options, see [Section 4.3.5, “Making a Partial Backup”](#) and [Section 5.1.4, “Table-Level Recovery \(TLR\)”](#).

- `--include-tables=REGEXP`

Property	Value
Command-Line Format	<code>--include-tables=REGEXP</code>
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`.

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.

`mysqlbackup` throws an error when no table matches the regular expression specified with `--include-tables`.

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`.

The option cannot be used together with the legacy `--include` option.

- `--exclude-tables=REGEXP`

Property	Value
Command-Line Format	<code>--exclude-tables=REGEXP</code>
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`.

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` 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 [Table 1.1, “Types of Files in a Backup”](#) 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 have the following extensions:

- `.ARM`: Archive storage engine metadata
- `.ARZ`: Archive storage engine data
- `.CSM`: CSV storage engine data
- `.CSV`: CSV storage engine data
- `.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

- `--only-innodb`

For back up only. Back up only InnoDB data and log files. 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.

Default: backups include files from all storage engines.

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

Property	Value
Command-Line Format	<code>--use-tts[={with-minimum-locking with-full-locking}]</code>
Type	Enumeration
Default Value	<code>with-minimum-locking</code>
Valid Values	<code>with-minimum-locking</code> <code>with-full-locking</code>

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:

- Individual partitions cannot be selectively backed up or restored. Tables selected by the `--include-tables` and `--exclude-tables` options are always backed up or restored in full.
- 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
- *For MySQL Enterprise Backup 8.0.20 and earlier:* Encrypted InnoDB tables are never included (a warning is issued in the log file whenever an encrypted InnoDB table that matches the table selection criteria has been skipped over).
- 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 5.1.5, “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 5.1.5, “Restoring Backups Created with the `--use-tts` Option”](#) apply. See [Section 5.1.5, “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 option of `--include`, which has been **deprecated**. For creating partial backups, use the `--include-tables` and `--exclude-tables` options instead.

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

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.

- `--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 5.1, “Performing a Restore Operation”](#) for details.

19.9 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`, `copy-back-and-apply-log`, `list-image`, and `extract` (not all of the options are applicable to all these commands though). For usage examples, see [Section 4.3.1, “Making a Single-File Backup”](#).

- `--backup-image=IMAGE`

See description of the option in [Section 19.4, “Backup Repository Options”](#)

- `--src-entry=STRING`

Property	Value
Command-Line Format	<code>--src-entry=STRING</code>
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` and `image-to-backup-dir` commands. 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, `f2`, 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`

Property	Value
Command-Line Format	<code>--dst-entry=PATH</code>
Type	Path name

Used with single-file backups to extract a 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`

Property	Value
Command-Line Format	<code>--sbt-database-name=NAME</code>
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 4.3.1.2, “Backing Up to Tape”](#) for usage details.

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

Property	Value
Command-Line Format	<code>--sbt-lib-path=PATH</code>
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 4.3.1.2, “Backing Up to Tape”](#) for usage details.

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

Property	Value
Command-Line Format	<code>--sbt-environment=VAR1=value1[,VAR2=value2[,...]]</code> 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.

19.10 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`

Property	Value
Command-Line Format	<code>--number-of-buffers=NUMBER</code>
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 13.1, “Optimizing Backup Performance”](#) and [Section 13.2, “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`

Property	Value
Command-Line Format	<code>--read-threads=NUMBER</code>
Type	Numeric
Default Value	1
Minimum Value	1
Maximum Value	15

Specifies the number of threads to use for reading data from disk. This option applies to these kinds of operations: `copy-back`, `copy-back-and-apply-log`, `extract`, `backup`, and `backup-and-apply-log`. 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 on backup directories and the apply-log phase of `copy-back-and-apply-log`, `backup-and-apply-log`, or `apply-incremental-backup`, the number of read threads is always 1 regardless of this option's setting. See [Section 13.1, “Optimizing Backup Performance”](#) and [Section 13.2, “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.

- `--process-threads=num_threads`

Property	Value
Command-Line Format	<code>--process-threads=NUMBER</code>
Type	Numeric
Default Value	6
Minimum Value	1
Maximum Value	15

Specifies the number of threads to use for processing data, including compression and uncompression, encryption and decryption, apply-log operations, and packing and extracting of backup images. For `backup-and-apply-log`, `copy-back-and-apply-log`, and `apply-incremental-backup`, `--process-threads` sets the worker threads number for the apply-log phase of the operation. The option is ignored for those operations that do not involve data processing like `copy-back` (unless decryption or uncompression is involved), `backup-dir-to-image`, or a backup operation that uses the `--incremental-with-redo-log-only` option.

Default: 6 for all operations to which the option is applicable.

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. See [Section 13.1, “Optimizing Backup Performance”](#) and [Section 13.2, “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`

Property	Value
Command-Line Format	<code>--write-threads=NUMBER</code>
Type	Numeric
Default Value	1
Minimum Value	1
Maximum Value	15

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 on backup directories and the apply-log phase of `copy-back-and-apply-log`, `backup-and-apply-log`, or `apply-incremental-backup`, the number of write threads is always 0 regardless of this option's setting. See [Section 13.1, “Optimizing Backup Performance”](#) and [Section 13.2, “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`

Property	Value
Command-Line Format	<code>--limit-memory=MB</code>
Type	Numeric
Default Value	100 for <code>apply-log</code> (without uncompression), 300 for other operations
Minimum Value	0
Maximum Value	999999
Unit	megabyte

Specify maximum memory in megabytes that can be used by `mysqlbackup`. It applies to all operations. Do not include any suffixes such as `mb` or `kb` in the option value.

Default: 100 for `apply-log` without uncompression, 300 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 300 MB limit, the maximum number of buffers is 18. If additional buffers are required because you increase the values for `--read-threads`, `--process-threads`, `--write-threads`, and/or `--number-of-buffers`, increase the `--limit-memory` value proportionally.

- `--sleep=MS`

Property	Value
Command-Line Format	<code>--sleep=MS</code>
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 all locking during backup (see [The Backup Process](#) for details). It can be used to back up a server with less disruption to normal database processing. There could be inconsistencies in both InnoDB and non-InnoDB data if any changes are made while those files are being backed up.

- `--lock-wait-timeout`

Property	Value
Command-Line Format	<code>--lock-wait-timeout</code>
Removed	8.0.16
Type	Numeric
Default Value	60
Minimum Value	1
Unit	second

For MySQL Enterprise Backup 8.0.16 and later: The option is no longer supported.

For MySQL Enterprise Backup 8.0.15 and earlier: 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`

Property	Value
Command-Line Format	<code>--page-reread-time=MS</code>
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 second).

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

Property	Value
Command-Line Format	<code>--page-reread-count=number</code>
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}`

Property	Value
Command-Line Format	<code>--on-disk-full=option</code>
Type	Enumeration
Default Value	abort

Property	Value
Valid Values	<code>abort</code> <code>warn</code> <code>abort_and_remove</code>

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 backups (full, incremental, compressed, partial (except for `TTS` backups), single-file, etc.). See [Table 1.1, “Types of Files in a Backup”](#), 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.

For MySQL Enterprise Backup 8.0.19 and later: The option makes binary log to be skipped not just for the current backup operation, but also for all subsequent incremental backups that are based on the current backup.

- `--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 backups (full, incremental, compressed, partial (except for `TTS` backups), single-file, etc.) of a replica server. See [Table 1.1, “Types of Files in a Backup”](#), 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.

- `--no-redo-log-archive`

For MySQL Enterprise Backup 8.0.17 and later: Skip using [redo log archiving](#) on the server during backup, which is used by default. The option has no effects for operations other than backups. See [Chapter 7, Backing up Using Redo Log Archiving](#) for details.

- `--skip-final-rescan`

For MySQL Enterprise Backup 8.0.16 and later: Skip the final rescan for InnoDB tables that have been modified by DDL operations, which is supposed to take place after `mysqlbackup` puts the database under a [backup lock](#) 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.

For MySQL Enterprise Backup 8.0.15 and earlier: Skip the final rescan for InnoDB tables that have been modified by DDL operations, which is supposed to take place 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.

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

Property	Value
Command-Line Format	<code>--optimistic-time=DATE-TIME</code>
Type	String
Default Value	<code>now</code>

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`, `YYYYMMDDHHMMSSYY-MM-DD`, `YYYY-MM-DD`, `YY-MM-DD`, or `HH.MM.SSYYYYMMDDTHHMMSS`, as designated by the ISO 8601 standard.

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`

Property	Value
Command-Line Format	<code>--optimistic-busy-tables=REGEXP</code>
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`

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.

19.11 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`

Property	Value
Command-Line Format	<code>--messages-logdir=PATH</code>
Type	Directory name
Default Value	<code>backup_dir/meta</code>

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
```

19.12 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}]`

Property	Value
Command-Line Format	<code>--show-progress[=destinations]</code>
Type	Enumeration
Valid Values	<code>stderr</code> <code>stdout</code> <code>file:FILENAME</code> <code>fifo:FIFONAME</code> <code>table</code> <code>variable</code>

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.

For MySQL Enterprise Backup 8.0.15 and earlier: 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 $(\text{orig_size} - \text{compressed_size}) / \text{orig_size}$. 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`

Property	Value
Command-Line Format	<code>--progress-interval=SECONDS</code>
Type	Numeric
Default Value	2
Minimum Value	1
Maximum Value	100000

Property	Value
Unit	<code>second</code>

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.

19.13 Encryption Options

These options are for creating encrypted single-file backups and for decrypting them. See [Chapter 10, 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`

Property	Value
Command-Line Format	<code>--key=KEY</code>
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 10, Encryption for Backups](#) on how to create a key. The option is incompatible with the `--key-file` option.

- `--key-file=PATH`

Property	Value
Command-Line Format	<code>--key-file=FILE</code>
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.

19.14 Options for Working with Encrypted InnoDB Tablespaces and Encrypted Binary/Relay Logs

MySQL Enterprise Backup supports encrypted InnoDB tablespaces and, for release 8.0.14 and later, encrypted binary/relay logs. For details on how MySQL Server encrypts and decrypts these items, see [InnoDB Data-at-Rest Encryption](#) and [Encrypting Binary Log Files and Relay Log Files](#). See [Chapter 6, Working with Encrypted InnoDB Tablespaces](#) and [Section 8.4, “Working with Encrypted Binary and Relay Logs”](#) on how `mysqlbackup` commands handle these encrypted items.

The following is the command-line option for working with encrypted InnoDB tables and binary/relay logs:

- `--encrypt-password[=STRING]`

Property	Value
Command-Line Format	<code>--encrypt-password=STRING</code>
Type	String

The user-supplied password by which `mysqlbackup` encrypts the master encryption key, which is used to encrypt the encryption keys for the InnoDB tablespaces or binary/relay log files.

The option must be used when backing up a server that has a keyring plugin enabled for InnoDB table or binary/relay log encryption and for restoring a backup containing encrypted InnoDB tables or binary/relay log. If the server is using the `keyring_encrypted_file` plugin, the password supplied with the option must match the value of the system variable `keyring_encrypted_file_password` on the server.

The same password supplied during backup must be supplied again during a `copy-back-and-apply-log`, `apply-log`, or an `apply-incremental-backup` operation for the backup, or `mysqlbackup` will error out when it encounters encrypted InnoDB tables or binary/relay logs during the operation. If different passwords were used for different backups in a sequence of full and incremental backups, make sure the very password used to create an individual backup is supplied when performing an `apply-log`, `apply-incremental-backup`, or `copy-back-and-apply-log` operation on it.

Users who do not want to supply the password on the command line or in a default file may use the option without specifying any value; `mysqlbackup` then asks the user to type in the password before the operation starts.

19.15 Cloud Storage Options

These options are for using cloud storage for single-file operations. See [Section 4.3.1.3, “Backing Up to Cloud Storage”](#), and [Section 5.1.6, “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).

- `--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`

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`

CA certificate directory, in addition to the system's default folder.

- `--cloud-buffer-size=MB`

Size of the buffer for cloud operations in megabytes. `mysqlbackup` accumulates data up to the size specified by this option before initiating a cloud transfer. The value has to be between 16 to 4096.

Default: 64

- Options used only for OpenStack Swift or compatible object storage services (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 by the Swift TempAuth identity system when the `--cloud-tempauth-url` option is used, by the OpenStack Keystone identity service when the `--cloud-identity-url` option is used, and by HTTP Basic Authentication when the `--cloud-basicauth-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 by the Swift TempAuth identity system when the `--cloud-tempauth-url` option is used, by the OpenStack Keystone identity service when the `--`

`cloud-identity-url` option is used, and by HTTP Basic Authentication when the `--cloud-basicauth-url` option is used.

- `--cloud-tempauth-url=SWIFT_TEMPAUTH-URL`

The TempAuth URL for authenticating user credentials.

- `--cloud-basicauth-url=SWIFT_BASICAUTH-URL`

The URL for HTTP Basic Authentication.

- `--cloud-storage-url=SWIFT_STORAGE-URL`

(For release 8.0.12 and later) The URL for Oracle Cloud Infrastructure Object Storage when using OAuth for client authentication.

- `--cloud-oauth-token=SWIFT_OAUTH-TOKEN`

(For release 8.0.12 and later) The OAuth token for Oracle Cloud Infrastructure Object Storage access. The option value must be in one of the following formats:

- `--cloud-oauth-token=oauth_token`, where `oauth_token` is the OAuth token itself
- `--cloud-oauth-token=file:oauth_filepath`, where `oauth_filepath` points to a file containing only the OAuth token.
- `--cloud-identity-url=SWIFT_KEYSTONE-URL`

The URL of the Keystone identity service, when it is used for authenticating user credentials.

- `--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.

- `--cloud-chunked-transfer=VALUE`

Use chunked transfer with Swift service.

If the option is set to `true`, backups are transferred and stored as Dynamic Large Objects (DLOs), for which multiple file segments are considered as a single file. Each backup larger than 5GB is

split into multiple parts with names in the form of `<object_name>_part_<number>`. See [the OpenStack documentation](#) for details.

If the option is set to false, `mysqlbackup` uploads the backup in segments in the size of the buffer, and the maximum number of segments a backup can have is determined by the Swift service.

Default: true



Warning

Set the option to false if chunked transfer is not supported by your cloud storage; otherwise, the `mysqlbackup` operation may fail.

One and only one of `--cloud-tempauth-url`, `--cloud-identity-url`, `--cloud-basicauth-url`, or `--cloud-storage-url` should be used when accessing a Swift service, or `mysqlbackup` will throw an error.

- 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.

19.16 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 8.1, “Setting Up a New replica”](#) for instructions.



Notes

- 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` option.
- For `TTS` backups for replica servers, use the `--slave-info` option to have the file `backup_gtid_executed.sql` generated and included in the backups.

- `--safe-slave-backup-timeout=SECONDS`

For MySQL Enterprise Backup 8.0.16 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 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.

For MySQL Enterprise Backup 8.0.15 and earlier: 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

For MySQL Enterprise Backup 8.0.16 and later: Do not set the value for this option too high. The wait time starts after all the InnoDB tables have been copied and ends when the read lock on non-InnoDB tables is asserted on the database. 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.

For MySQL Enterprise Backup 8.0.15 and earlier: Do not set the value for this option too high. The wait time starts after all the InnoDB tables have been copied and ends when the global read lock is asserted on the database. 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 a lot of tables will need to be copied again during the lock period, resulting in a long lock time. In such a case, it would have been more efficient to have let `mysqlbackup` timeout and then restart the backup operation, so the modified tables are copied without a lock. It is therefore not helpful to set a high timeout value for the option.

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.

For MySQL Enterprise Backup 8.0.16 and later: All non-InnoDB tables are locked before suspending, putting them into a read-only state, unless you turn off locking with the `--no-locking`. The `--only-innodb` option also prevents the locking step. You can also use a combination of `--only-innodb` and `--suspend-at-end` to back up only certain InnoDB tables.

For MySQL Enterprise Backup 8.0.15 and earlier: All tables are locked before suspending, putting the database into a read-only state, unless you turn off locking with the `--no-locking`. The `--only-innodb` option also prevents 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 ..."`

Property	Value
Command-Line Format	<code>--exec-when-locked="utility arg1 arg2 ..."</code>
Type	String

For MySQL Enterprise Backup 8.0.16 and later: The specified `utility` is executed when all non-InnoDB tables are locked near the end of a backup operation.

For MySQL Enterprise Backup 8.0.15 and earlier: The specified `utility` is executed when all tables are locked near the end of a backup operation.

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_com
```

Or on Windows systems:

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

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.

19.17 Other Options

This section explains any options that are not covered by other sections in this chapter.

- `--error-code=CODE`

Property	Value
Command-Line Format	<code>--error-code</code>
Type	Numeric
Minimum Value	0
Maximum Value	19

Specifies the exit code for which the `print-message` command prints the corresponding exit message. See [Section 16.1](#), “Exit codes of MySQL Enterprise Backup” for details.

Chapter 20 Configuration Files and Parameters

You can specify `mysqlbackup` options either on the command line or as configuration parameters inside a configuration file.

`mysqlbackup` looks for and reads MySQL configuration files as `mysqld` does (see explanations in [Using Option Files](#)). You can also supply a configuration file to `mysqlbackup` using the `--defaults-file` option. In general, `mysqlbackup` follows the `mysql` style of processing configuration options: `[mysqlbackup]` and `[client]` group options listed in a configuration file are passed as command-line options. Any command-line options that you specify when you run `mysqlbackup` override the values from the configuration file. In the case of duplicate options, the last instance takes precedence. `mysqlbackup` also reads options in the `[mysqld]` group in the configuration file to detect parameters related to the source repository when it is not connected to `mysqld` (for example, when restoring a [non-TTS backup](#)).

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.

Server Data Locations and Options Files

`mysqlbackup` reads the locations of the MySQL data (data files, logs, etc.) to be backed up or restored from the following sources:

- For backup operations and partial restore operations, the information is retrieved from `mysqld`.
- For non-partial restore operations, the information is supplied to `mysqlbackup` as parameters through:
 - The `mysqlbackup` command line, as command-line options.
 - A configuration file (see explanation above at the beginning of the chapter). The parameters are read first under the `[mysqlbackup]` group, then under the `[client]` group, and finally under the `[mysqld]` group. You can put common connection parameters used for restore operations (for example, user login, host name, etc.) into the configuration file.

Configuration Files Stored Inside the Backups

Each set of backup data includes a configuration file, `backup-my.cnf`, containing a set of configuration parameters. The `mysqlbackup` command generates this file to record the settings that apply to the backup data. Here is a sample `backup-my.cnf` file generated by `mysqlbackup`:

Example 20.1 Sample `backup-my.cnf` file

```
#
# Generated backup-my.cnf file.
# Auto generated by mysqlbackup program.
#
[mysqld]
innodb_data_file_path=ibdata1:12M:autoextend
innodb_log_file_size=50331648
innodb_log_files_in_group=2
innodb_page_size=16384
innodb_checksum_algorithm=crc32
innodb_buffer_pool_filename=ib_buffer_pool
innodb_undo_tablespace=2
```

All file paths contained in the generated `backup-my.cnf` are relative to the data directory under the [backup directory](#).

These configuration parameters are read by `mysqlbackup` during operations like `apply-log`, in which the parameters are read from this file to determine how the backup data is structured. These parameters can also be used in a restore to compare the InnoDB settings of the target server with those of the backed-up server, so that any necessary adjustments can be made; see [Starting the Restored Server](#) for details. Only the minimally-required parameters are stored in `backup-my.cnf`: for example, the `innodb_data_home_dir` and `innodb_log_group_home_dir` options are omitted from the `backup-my.cnf` file when they just point to the data directory under the [backup directory](#) (`backup-dir/datadir` usually).

Part IV Appendixes

Table of Contents

A Frequently Asked Questions for MySQL Enterprise Backup	181
B Limitations of MySQL Enterprise Backup	183
C Compatibility Information for MySQL Enterprise Backup	185
C.1 Cross-Platform Compatibility	185
C.2 Compatibility with MySQL Versions	185
C.3 Compatibility with Older MySQL Enterprise Backup	185
D Backup History Table Update	187
E SBT Backup History Table Update	189
F Backup Progress Table Update	191
MySQL Enterprise Backup Glossary	193

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:](#) What versions of the MySQL server does MySQL Enterprise Backup 8.0.21 support?
- [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 or apply-incremental-backup 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?
- [A.8:](#) What would happen if I start a server directly using a raw directory backup, without running either the copy-back or the apply-log operation?

Questions and Answers

A.1: What versions of the MySQL server does MySQL Enterprise Backup 8.0.21 support?

See [Section C.2, “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 as a database operates, 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; the tables can then be individually included or excluded from backups, and disk space is freed when the tables are dropped or truncated.
- 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 create a dump of your entire server instance, then turn on `innodb_file_per_table` before re-creating the database, 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` or `apply-incremental-backup` 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 going after `mysqlbackup`. 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.1, “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.

A.8: What would happen if I start a server directly using a raw directory backup, without running either the `copy-back` or the `apply-log` operation?

This should never be attempted. Not only would the server crash, but the backup would likely get corrupted and become unusable. This is because the directory backup contains metadata created by `mysqlbackup` that the MySQL server would not understand; also, the raw backup might be inconsistent and need to be brought up-to-date by an [apply-log operation](#), so that changes made to the database during the backup process can be applied.

Appendix B Limitations of MySQL Enterprise Backup

Please refer to the [MySQL Enterprise Backup 8.0 Release Notes](#) for a list of fixed bugs for `mysqlbackup`. Here is a list of limitations of MySQL Enterprise Backup:

- In Linux, Unix, and OS X systems, `mysqlbackup` does not record file ownership or permissions of the files that are backed up. Upon restore, these files might have different ownership (for example, being owned by `root` now rather than `mysql`). They might also have different read/write permissions (for example, being readable by anyone rather than just the file owner). When planning your backup strategy, survey the files in the MySQL data directory to ensure they have consistent owner and permission settings. When executing a restore operation, use an appropriate combination of `su`, `umask`, `chown`, and `chmod` on the restored files to set up the same owners and privileges as on the original files. The simplest way to ensure correct file ownership and permissions is to run the restore operation as the same user who runs the server, typically `mysql`.
- 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.
- 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.
- 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.
- Backup of a partitioned table using [transportable tablespace \(TTS\)](#) would fail when any (or all) of its partitions were created in a shared tablespace.
- Restoring a partitioned table backed up using [transportable tablespace \(TTS\)](#), even when the `--force` option is used, would fail if any of the partitions was created outside of the backed-up server's data directory.
- If a table containing full-text search (FTS) index is backed up using [transportable tablespace \(TTS\)](#), after it is restored, the FTS index will be corrupted. Users will need to recreate the index with the following command:

```
mysql> ALTER TABLE mytable ENGINE = INNODB;
```

Then, check that there are no more errors with the table:

```
mysql> CHECK TABLE mytable;
```

- Tables created on the MySQL server with the `ANSI_QUOTES` SQL mode cannot be backed up using [transportable tablespace \(TTS\)](#).

-
- MySQL Enterprise Backup does not include the `.pem` files from the server into the backup. The files are part of the server instance when SSL connections are enabled.
 - During a backup process, if a `CREATE INDEX` statement with `ALGORITHM = INPLACE` is issued when the backup process is going on, because the statement will not go into the redo log of the MySQL server (see [Sorted Index Builds](#) for details), it cannot be recorded in the backup, and the index will not be recreated by `mysqlbackup` when the backup is restored.
 - 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.
 - 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.
 - Some limitations apply when `mysqlbackup` works with [encrypted InnoDB tables](#). See [the discussion here](#) for details.
 - Backup operations fail if the server has been started with `--innodb_undo_log_encrypt=ON`
 - Backup operations may fail if checksums for redo log pages are disabled (i.e., if `--innodb_log_checksums` is `OFF` or `FALSE` or `0`) on the server.
 - *For MySQL Enterprise Backup 8.0.19 and later:* It is safe to have DDL operations (`CREATE TABLE`, `RENAME TABLE`, `DROP TABLE`, `ALTER TABLE`, and operations that map to `ALTER TABLE` like `CREATE INDEX`) happening on the server in parallel with a backup operation as long as:
 - The tables involved exist in their own tablespaces, instead of being in the system tablespace or some general tablespaces.
 - These server features have not been applied to the tables involved:
 - [Data-at-rest encryption](#)
 - [Page-level compression](#)
 - [Full-text indexing](#)
 - The backup is not taken with the following `mysqlbackup` features:
 - [Optimistic backup](#)
 - [Transportable tablespace \(TTS\)](#)
 - [Redo log archiving](#)
 - [Incremental backups with-redo-log-only](#)
 - [Incremental backup using page-tracking](#)

Appendix C Compatibility Information for MySQL Enterprise Backup

Table of Contents

C.1 Cross-Platform Compatibility	185
C.2 Compatibility with MySQL Versions	185
C.3 Compatibility with Older MySQL Enterprise Backup	185

This section describes information related to compatibility issues for MySQL Enterprise Backup releases.

C.1 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.2 Compatibility with MySQL Versions

MySQL Enterprise Backup 8.0.21 only supports MySQL 8.0.21.

For earlier versions of MySQL 8.0, use the MySQL Enterprise Backup version with the same version number as the server.

For MySQL 5.7, use MySQL Enterprise Backup 4.1.

For MySQL 5.6, use MySQL Enterprise Backup 3.12.

C.3 Compatibility with Older MySQL Enterprise Backup

MySQL Enterprise Backup 8.0.21 is incompatible with earlier MySQL Enterprise Backup versions—it does not work with backups created with them.

Appendix D Backup History Table Update

The `mysql.backup_history` table has been updated with the release of MySQL Enterprise Backup 8.0.12 in the following ways:

- Changed the storage engine from CSV to InnoDB
- Added a new column for server UUIDs

When MySQL Enterprise Backup 8.0.12 or later tries to perform its first full backup on a database, it automatically checks the format of the `mysql.backup_history` table. If it detects that the table is in the old format (which means the server has been upgraded from 8.0.11 or earlier (or 5.7.22 or earlier) and has been backed up by MySQL Enterprise Backup before), it attempts to perform a format update on the table automatically with the following steps:

1. Create in the new format a table named `mysql.backup_history_new` and copy into it data from the original `mysql.backup_history` table.
2. Rename the original `mysql.backup_history` table to `mysql.backup_history_old`, and the `mysql.backup_history_new` table to `mysql.backup_history`.
3. *For MySQL Enterprise Backup 8.0.21 and later:* Drop the `mysql.backup_history_old` table.

For the migration to the new table format to succeed, before performing the first backup task with MySQL Enterprise Backup 8.0.12 or later for the first time on a MySQL Server that has been upgraded from 8.0.11 or earlier and has been backed up by MySQL Enterprise Backup before, grant the required privileges to the `mysqlbackup` user on the server by issuing these statements at the `mysql` client :

```
GRANT CREATE, INSERT, DROP ON mysql.backup_history_old TO 'mysqlbackup'@'localhost';
GRANT CREATE, INSERT, DROP, ALTER ON mysql.backup_history_new TO 'mysqlbackup'@'localhost';
```

If these privileges are not granted, the first full backup on the upgraded server will fail with an error message similar to the following:

```
180612 08:40:45 MAIN ERROR: MySQL query 'DROP TABLE IF EXISTS mysql.backup_history_old': 1142, DROP com
mysqlbackup failed with errors!
```

These privileges are no longer needed after the first full backup has been performed by MySQL Enterprise Backup 8.0.12 or later, by which point they can be revoked.



Note

If you are working with a multiprimary Group Replication setting, make sure these privileges are granted on all primary nodes; see also [Chapter 9, Using MySQL Enterprise Backup with Group Replication](#).

Appendix E SBT Backup History Table Update

The `mysql.backup_sbt_history` table has been updated with the release of MySQL Enterprise Backup 8.0.21 in the following ways:

- Changed the storage engine from CSV to InnoDB
- Added a new auto-increment primary key column `id`

When MySQL Enterprise Backup 8.0.21 or later tries to perform its first full backup on a database using the SBT API (see [Section 11.1, “Backing Up to Tape with Oracle Secure Backup”](#) for details), it automatically checks the format of the `mysql.backup_sbt_history` table. If it detects that the table is in the old format (which means the server has been upgraded from 8.0.20 or earlier and has been backed up by MySQL Enterprise Backup before using the SBT API), it attempts to perform an update on the table automatically with the following steps:

1. Create in the new format a table named `mysql.backup_sbt_history_new` and copy into it data from the original `mysql.backup_sbt_history` table.
2. Rename the original `mysql.backup_sbt_history` table to `mysql.backup_sbt_history_old`, and the `mysql.backup_sbt_history_new` table to `mysql.backup_sbt_history`.
3. Drop the `mysql.backup_history_old` table.

For the migration to the new table format to succeed, before performing the first backup task using the SBT API with MySQL Enterprise Backup 8.0.21 or later on a MySQL Server that has been upgraded from 8.0.20 or earlier and has been backed up by MySQL Enterprise Backup before with the SBT API, grant the required privileges to the `mysqlbackup` user on the server by issuing these statements at the `mysql` client :

```
GRANT ALTER ON mysql.backup_sbt_history TO 'mysqlbackup'@'localhost';
GRANT CREATE, INSERT, DROP ON mysql.backup_sbt_history_old TO 'mysqlbackup'@'localhost';
GRANT CREATE, INSERT, DROP, ALTER ON mysql.backup_sbt_history_new TO 'mysqlbackup'@'localhost';
```

If these privileges are not granted, the first full backup on the upgraded server using the SBT API will fail with an error message similar to the following:

```
200612 08:40:45 MAIN ERROR: MySQL query 'DROP TABLE IF EXISTS mysql.backup_sbt_history_old': 1142, DROP
mysqlbackup failed with errors!
```

These privileges are no longer needed after the first full backup with SBT API has been performed by MySQL Enterprise Backup 8.0.21 or later, by which point they can be revoked.



Note

If you are working with a multiprimary Group Replication setting, make sure these privileges are granted on all primary nodes; see also [Chapter 9, Using MySQL Enterprise Backup with Group Replication](#).

Appendix F Backup Progress Table Update

The `mysql.backup_progress` table has been updated with the release of MySQL Enterprise Backup 8.0.19 in the following ways:

- Changed the storage engine from CSV to InnoDB
- Added a new auto-increment primary key column `id`
- Added a composite index on the `backup_id` and `current_timestamp` columns

When MySQL Enterprise Backup 8.0.19 or later tries to perform its first full backup on a database, it automatically checks the format of the `mysql.backup_progress` table. If it detects that the table is in the old format (which means the server has been upgraded from 8.0.18 or earlier and has been backed up by MySQL Enterprise Backup before), it attempts to perform a format update on the table automatically with the following steps:

1. Create in the new format a table named `mysql.backup_progress_new` and copy into it data from the original `mysql.backup_progress` table.
2. Rename the original `mysql.backup_progress` table to `mysql.backup_progress_old`, and the `mysql.backup_progress_new` table to `mysql.backup_progress`.
3. *For MySQL Enterprise Backup 8.0.21 and later:* Drop the `mysql.backup_progress_old` table.

For the migration to the new table format to succeed, before performing the first backup task with MySQL Enterprise Backup 8.0.19 or later for the first time on a MySQL Server that has been upgraded from 8.0.18 or earlier and has been backed up by MySQL Enterprise Backup before, grant the required privileges to the `mysqlbackup` user on the server by issuing these statements at the `mysql` client: :

```
GRANT ALTER ON mysql.backup_progress TO 'mysqlbackup'@'localhost';
GRANT CREATE, INSERT, DROP ON mysql.backup_progress_old TO 'mysqlbackup'@'localhost';
GRANT CREATE, INSERT, DROP, ALTER ON mysql.backup_progress_new TO 'mysqlbackup'@'localhost';
```

If these privileges are not granted, the table upgrade will fail with an error message similar to the following:

```
191219 20:48:43 MAIN ERROR: MySQL query 'RENAME table mysql.backup_progress TO mysql.backup_progress_new TO mysql.backup_progress': 1142, ALTER command denied to user 'mysqlbackup'@'localhost' for table 'backup_progress'
```

These privileges are no longer needed after the first full backup has been performed by MySQL Enterprise Backup 8.0.19 or later, by which point they can be revoked.



Note

If you are working with a multiprimary Group Replication setting, make sure these privileges are granted on all primary nodes; see also [Chapter 9, Using MySQL Enterprise Backup with Group Replication](#).

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.
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 **slave** 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. 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.

See Also [Antelope](#), [file format](#), [MySQL Enterprise Backup](#), [row format](#), [system tablespace](#).

binary log

A file containing a record of all statements that attempt to change table data. These statements can be replayed to bring slave servers up to date in a **replication** scenario, or to bring a database up to date after restoring table data from a backup. The binary logging feature can be turned on and off, although Oracle recommends always enabling it if you use replication or perform backups.

You can examine the contents of the binary log, or replay those statements during replication or recovery, by using the `mysqlbinlog` command. For full information about the binary log, see [The Binary Log](#). For MySQL configuration options related to the binary log, see [Binary Logging Options and Variables](#).

For the **MySQL Enterprise Backup** product, the file name of the binary log and the current position within the file are important details. To record this information for the master server when taking a backup in a replication context, you can specify the `--slave-info` option.

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.

MySQL Enterprise Backup 8.0 does not support cold backups.

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 OS X systems, and **my.ini** on Windows systems. The **MySQL Enterprise Backup** stores its default configuration settings in this file, under a [\[mysqlbackup\]](#) section. For convenience, MySQL Enterprise Backup can also read settings from the [\[client\]](#) section, for configuration options that are common between MySQL Enterprise Backup and other programs that connect to the MySQL server.

See Also [my.cnf](#), [my.ini](#).

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.

Cold backups are not supported by **MySQL Enterprise Backup** 8.0. Therefore a connection to the server is always needed for **MySQL Enterprise Backup** 8.0 to back it up.

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 4.3.3, “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 and minimizing the duration of the **suspend** stage where you run customized backup logic while the MySQL server is **locked**.

See Also [cold backup](#), [hot backup](#), [InnoDB](#), [locking](#), [replication](#), [slave](#), [suspend](#).

E

exclude

In a **partial backup**, to select a set of tables, databases, or a combination of both to be omitted from the backup. Contrast with **include**.

See Also [partial backup](#).

extract

The operation that retrieves some content from an **image** file produced by a **single-file backup**. It can apply to a single file (unpacked to an arbitrary location) or to the entire backup (reproducing the original directory structure of the backup data). These two kinds of extraction are performed by the [mysqlbackup](#) options [extract](#) and [image-to-backup-dir](#), respectively.

See Also [image](#), [single-file backup](#).

F

.frm file

A file containing the metadata, such as the table definition, of a MySQL table. [.frm](#) files were removed in MySQL 8.0 but are still used in earlier MySQL releases. In MySQL 8.0, data previously stored in [.frm](#) files is stored in **data dictionary** tables.

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 the associated [.sdi](#) 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 4.3.3, “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 OS X systems.
See Also [configuration file](#), [my.ini](#).

my.ini

The typical name for the MySQL **configuration file** on Windows systems.
See Also [configuration file](#), [my.cnf](#).

MyISAM

A MySQL storage engine, formerly the default for new tables. In MySQL 5.5 and higher, **InnoDB** becomes the default storage engine. MySQL Enterprise Backup can back up both types of tables, and tables from other storage engines also. The backup process for InnoDB tables (**hot backup**) is less disruptive to database operations than for MyISAM tables (**warm backup**).
See Also [hot backup](#), [InnoDB](#), [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 cannot perform a **cold backup** with **MySQL Enterprise Backup** 8.0. 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 (**MySQL Enterprise Backup** 8.0 does not support **cold backups**). 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.

optimistic Incremental Backup

In an optimistic incremental backup `mysqlbackup` scans InnoDB data files that have been modified since the last backup for changed pages and then saves them into the incremental backup. It is performed by specifying `--incremental=optimistic`. See [Full-scan versus Optimistic Incremental 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 1.2, “Overview of Backup Types”](#) for an overview, [Section 19.10, “Performance / Scalability / Capacity Options”](#) for the relevant `mysqlbackup` options, and [Chapter 13, 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 **slave** server for the events read from the **binary log** of the master and written by the slave 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 Replica 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 master. 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 **master** server synchronized with a set of **slave** servers. With **MySQL Enterprise Backup**, you might take a backup on one server, and restore on a different system to create a new slave server with the data already in place. You might also back up data from a slave server rather than the master, to minimize any slowdown of the overall system.

See Also [master](#), [slave](#).

repository

We distinguish between the **server repository** and the **backup repository**.

See Also [backup repository](#), [server repository](#).

restore

The converse of the **backup** operation. The data files from a **prepared backup** are put back into place to repair a data issue or bring the system back to an earlier state.

See Also [backup](#), [prepared backup](#).

row format

The disk storage format for a row from an InnoDB table. As InnoDB gains new capabilities such as compression, new row formats are introduced to support the resulting improvements in storage efficiency and performance.

Each table has its own row format, specified through the [ROW_FORMAT](#) option. To see the row format for each InnoDB table, issue the command [SHOW TABLE STATUS](#). Because all the tables in the system tablespace share the same row format, to take advantage of other row formats typically requires setting the [innodb_file_per_table](#) option, so that each table is stored in a separate tablespace.

S

.sdi file

A file containing the metadata (referred to as [Serialized Dictionary Information \(SDI\)](#)) of a MyISAM table.

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.

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](#).

Table-Level Recovery (TLR)

Table-Level Recovery (TLR) is a selective restore of tables or schemas from a backup; see [Section 5.1.4](#), “[Table-Level Recovery \(TLR\)](#)” for details.

See Also [partial restore](#).

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](#).

TLR

Short form for Table-Level Recovery.

See Also [partial restore](#), [Table-Level Recovery \(TLR\)](#).

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 5.1.5, “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

--log-bin option, 137
--relay-log option, 137

A

Antelope, 117, 193
apply, 193
apply-incremental-backup option, 121
--apply-log option, 121
.ARM file, 193
.ARZ file, 193

B

backup, 193
backup directory, 140, 193
backup option, 120
backup repository, 193
backup-and-apply-log option, 120
--backup-dir option, 140
backup-dir-to-image option, 125
backup-image option, 140
backup-my.cnf, 193
backup-my.cnf file, 7
backup-to-image option, 120, 152
backups
 cold, 5
 compressed, 5, 57, 60, 71, 143
 containing encrypted InnoDB tablespaces, 75
 controlling overhead, performance, and scalability, 155
 differential, 51
 encrypted, 89
 full, 50
 hot, 5
 incremental, 5, 51, 145
 InnoDB tables only, 117
 logical, 5
 message logging, 161
 optimistic, 60
 parallel, 5
 partial, 57, 148
 physical, 5
 prepared, 7, 71
 preparing to restore, 71
 progress report, 163
 raw, 7, 71
 scheduled, 63
 single-file, 5, 46
 streaming, 5, 48
 to cloud, 48
 to tape, 48
 with Oracle Secure Backup, 91
 troubleshooting, 107
 uncompressed, 5, 59
 verifying, 42

 warm, 5
backup_content.xml, 7
backup_content.xml file, 111
backup_create.xml, 7
backup_create.xml file, 111
backup_history table, 109
 update, 187
backup_progress table, 109
 update, 191
backup_sbt_history table
 update, 189
backup_variables.txt file, 7
Barracuda, 117, 194
benchmarking, 97
binary log, 72, 194
binlog, 194
.bz file, 193

C

cloud backups, 48
--cloud-access-key-id option, 170
--cloud-aws-region option, 170
--cloud-basicauth-url, 169
--cloud-bucket option, 170
--cloud-buffer-size option, 168
--cloud-ca-info option, 168
--cloud-ca-path option, 168
--cloud-chunked-transfer, 169
--cloud-identity-url, 169
cloud-oauth-token, 169
--cloud-object, 168
--cloud-object-key option, 170
--cloud-password, 168
--cloud-proxy option, 167
--cloud-region, 169
--cloud-secret-access-key option, 170
--cloud-service option, 167
cloud-storage-url, 169
--cloud-tempauth-url, 169
--cloud-tenant, 169
--cloud-trace option, 167
--cloud-user-id, 168
cold backup, 5, 194
command-line tools, 5
commands, mysqlbackup, 119
--comments option, 143
--comments-file option, 143
comments.txt file, 7, 143
--compress option, 57, 143
--compress-level option, 57, 145
--compress-method option, 144
compressed backups, 5, 57, 60, 71, 143
compression, 194
compression level, 194
configuration file, 195
configuration options, 175
connection, 195
connection options, 136

- copy-back option, 65, 123
- copy-back-and-apply-log option, 43, 122
- corruption problems, 108
- counter-container, 168
- crash recovery, 71, 195
- cron jobs, 63
- .CSM file, 7
- .CSV file, 7

D

- data dictionary, 195
- database, 195
- datadir directory, 7
- datadir option, 137
- data_home_dir option, 138
- decrypt option, 166
- decryption, 89
- differential backup, 195
- disable-manifest option, 155
- disk storage for backup data, 5, 48
- distributed file system, 64
- downtime, 195
- dst-entry option, 153

E

- encrypt option, 166
- encrypt-password option, 167
- encrypted backups, 89
- encrypted InnoDB tables, 75
- encrypted InnoDB tablespaces, 75
- encrypted undo tablespaces, 75
- encryption, 89
- error-code option, 107, 173
- exclude, 196
- exclude-tables option, 57, 149
- exec-when-locked option, 172
- exit codes, 107, 173
- extract, 196
- extract option, 126, 152

F

- FAQ, 181
- file format, 196
- files backed up, 7
- frequently asked questions, 181
- .frm file, 196
- full backup, 50, 196
- full-scan incremental backup, 54

G

- GRANT statement, 36
- group replication, 87

H

- history table, 196
- hot backup, 5, 196

I

- ibbackup_logfile file, 7
- .ibd file, 7, 197
- ibdata file, 7, 197
- ibreset command, 108
- .bz file, 7
- .ibz file, 7, 197
- ib_logfile file, 7
- image, 197
- image-to-backup-dir option, 125, 152, 152
- image_files.xml file, 7, 111
- include, 197
- include option, 57, 152
- include-tables option, 57, 148
- incremental backup, 5, 145, 197
 - full scan versus optimistic, 54
 - using page tracking, 53
- incremental option, 146
- incremental-backup-dir option, 148
- incremental-base option, 147
- incremental-with-redo-log-only option, 146
- InnoDB, 197
- InnoDB tables, 5, 7, 117, 117
 - compressed backup feature, 57
 - incremental backup feature, 51
- installing MySQL Enterprise Backup, 23
- instance, 197

K

- key option, 166
- key-file option, 166

L

- limit-memory option, 157
- list-image option, 125, 152
- lock-wait-timeout option, 158
- locking, 198
- log, 7, 120, 198
- log-bin-index, 138
- logical backup, 5, 198
- logs
 - of backup operations, 109, 187, 189, 191
- LSN, 51, 145, 198

M

- manifest, 7, 110, 155, 198
- media management software, 198
- media management software (MMS) products, 91
- MEMORY tables, 63
- message logging, 161
- meta directory, 7
- MMS products, 91
- monitoring backup jobs, 103
- .MRG file, 198
- my.cnf, 199
- my.ini, 199
- .MYD file, 7

- .MYD file, 198
- .MYI file, 7
- .MYI file, 198
- .sdi file, 7
- MyISAM, 199
- MyISAM tables, 117
- MySQL Enterprise Backup, 199
- mysqlbackup, 117, 199
 - and media management software (MMS) products, 91
 - commands, 119
 - configuration options, 175
 - examples, 46
 - files produced, 7
 - modes of operation, 119
 - options, 129
 - overview, 5
 - required privileges, 36
 - using, 35
- mysqlbinlog command, 72
- mysqldump, 63, 199

N

- no-history-logging option, 143
- no-locking option, 157
- non-TTS backup, 199
- number-of-buffers option, 155

O

- offline, 199
- on-disk-full option, 159
- online, 199
- only-innodb option, 149
- rename option, 151
- only-known-file-types option, 149
- .opt file, 7, 199
- optimistic backup, 60, 160, 161, 200
- optimistic incremental backup, 54, 146
- optimistic Incremental Backup, 200
- optimistic-busy-tables, 161
- optimistic-time, 160
- options, mysqlbackup, 129
 - connection, 136
 - for cloud storage, 167
 - for compression, 143
 - for controlling backup overhead, performance, and scalability, 155
 - for controlling message logging, 161
 - for controlling progress reporting, 163
 - for encrypted InnoDB tablespaces, 166
 - for encryption, 166
 - for generating metadata, 142
 - for incremental backups, 145
 - for partial backups, 148
 - for single-file backups, 152
 - for special types of backups, 170
 - in configuration files, 175

- layout of backup files, 139
- layout of database files, 137
- options in common with mysql, 134
- other, 173
- standard options, 134
- Oracle Secure Backup, 200
- OSB, 200
- other operations, 126

P

- page-reread-count option, 158
- page-reread-time option, 158
- .par file, 200
- parallel backup, 97, 100, 200
- parallel backups, 5
- partial backup, 57, 148, 200
- partial restore, 68, 200
- performance
 - of backups, 97
 - of restores, 100
- performance of backup operations, 5
- physical backup, 5, 200
- point in time, 200
- point-in-time recovery, 72
- posix_fadvise() system call, 5
- prepared backup, 7, 71, 201
- print-message option, 126
- privileges, 36
- process-threads option, 156
- progress indicator, 163
- progress table, 201
- progress-interval, 166

R

- RAID, 97, 100
- raw backup, 7, 71, 201
- read-threads option, 155
- redo log, 201
- redo log archiving, 79
- regular expression, 201
- relay log, 201
- relay-log-index, 138
- free-os-buffers, 161
- replica, 81, 83, 201
- replication, 81, 83, 83, 201
- repository, 202
- restore, 202
- restoring a backup, 65
 - at original location, 43
 - backup created with the --use-tts option, 69
 - examples, 65
 - mysqlbackup options, 121
 - partial restore, 68
 - point-in-time recovery, 71
 - preparation, 71
 - restore external tablespaces at different locations, 70
- row format, 202

S

- safe-slave-backup-timeout, 171
- SBT, 202
 - backup history table update, 189
- sbt-database-name option, 154
- sbt-environment option, 154
- sbt-lib-path option, 154
- .sdi file, 202
- selective backup, 202
- selective restore, 202
- server, 202
- server repository, 202
- show-progress, 163
- single-file backup, 5, 46, 124, 152, 202
- skip-binlog, 159
- skip-final-rescan, 160
- no-redo-log-archive, 160
- skip-relaylog, 160
- skip-unused-pages, 159
- slave-info option, 170
- sleep option, 157
- source, 83, 202
- space for backup data, 5
- src-entry option, 153
- start-lsn option, 147
- storage access network, 64
- streaming, 48, 203
- streaming backups, 5
- suspend, 203
- suspend-at-end option, 172
- system backup to tape, 203
- system tablespace, 7, 203

T

- table, 203
- Table-Level Recovery (TLR), 68, 203
- tablespace, 203
- tablespace tracker file, 18
- tape backups, 48, 91
- TLR, 203
- transportable tablespace, 203
- .TRG file, 203
- troubleshooting for backups, 107
- TTS, 204
- TTS backup, 204

U

- uncompress option, 145
- uncompressed backups, 5, 59
- use-tts option, 150
- using with MySQL Enterprise Firewall, 105

V

- validate option, 124
- validating a backup, 124
- verifying a backup, 42

W

- warm backup, 5, 204
- what is new, 25
- with-timestamp option, 142
- write-threads option, 156