MySQL Enterprise Backup User's Guide (Version 26.10.0)

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 26.10 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: 2026-10-19 (revision: 8791)

Table of Contents

Preface and Legal Notices ... xi
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 26.10? ... 25
II Using MySQL Enterprise Backup ... 27
4 Backing Up a Database Server ... 31
4.1 Before the First Backup ... 31
4.1.1 Collect Database Information ... 31
4.1.2 Grant MySQL Privileges to Backup Administrator ... 32
4.1.3 Designate a Location for the Backup Directory ... 36
4.2 The Typical Backup / Verify / Restore Cycle ... 36
4.2.1 OS User for Running mysqlbackup ... 36
4.2.2 Backing Up an Entire MySQL Instance ... 37
4.2.3 Verifying a Backup ... 40
4.2.4 Restoring a Database ... 41
4.3 Backup Scenarios and Examples ... 44
4.3.1 Making a Single-File Backup ... 44
4.3.2 Making a Full Backup ... 49
4.3.3 Making a Differential or Incremental Backup ... 50
4.3.4 Making a Compressed Backup ... 56
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 Server ... 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 `--use-tts` Option ... 69
5.1.6 Restoring External InnoDB Tablespaces to Different Locations ... 70
5.1.7 Advanced: Preparing and Restoring a Directory Backup ... 70
5.2 Restoring a Backup from Cloud Storage to a MySQL Server ... 70
5.3 Point-in-Time Recovery ... 71
5.4 Restoring a Backup with a Database Server 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 ... 84
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 Using MySQL Enterprise Backup with MySQL Enterprise Firewall ... 103
15 Using LDAP for Server Authentication ... 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 ... 111
III `mysqlbackup` Command Reference ... 113
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 ... 123
18.5 Other Single-File Backup Operations ... 124
18.6 Other Operations ... 126
19 `mysqlbackup` Command-Line Options ... 129
19.1 General Options ... 134
19.2 Connection Options ... 136
19.3 Server Repository Options ... 137
19.4 Backup Repository Options ... 140
19.5 Metadata Options ... 144
19.6 Compression Options ... 144
19.7 Incremental Backup Options ... 147
19.8 Partial Backup and Restore Options ... 149
19.9 Single-File Backup Options ... 154
19.10 Performance / Scalability / Capacity Options ... 156
19.11 Message Logging Options ... 163
19.12 Progress Report Options ... 164
19.13 Encryption Options ... 168
19.14 Options for Working with Encrypted InnoDB Tablespaces and Encrypted Binary/ Relay Logs ... 168
19.15 Cloud Storage Options ... 169
19.16 Options for Special Backup Types ... 174
20 Configuration Files and Parameters ... 179
IV Appendixes ... 181
A Frequently Asked Questions for MySQL Enterprise Backup ... 185
B Limitations of MySQL Enterprise Backup ... 187
C Compatibility Information for MySQL Enterprise Backup ... 191
C.1 Supported Platforms ... 191
C.2 Cross-Platform Compatibility ... 191
C.3 Compatibility with MySQL Versions ... 191
C.4 Compatibility with Older MySQL Enterprise Backup ... 192

D Backup History Table Update 193
E SBT Backup History Table Update 195
F Backup Progress Table Update 197
MySQL Enterprise Backup Glossary 199
Index 209

List of Tables

1.1 Types of Files in a Backup 7
4.1 Information Needed to Back Up a Database 31
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 44
4.2 Single-File Backup to Relative Path 44
4.3 Single-File Backup to Standard Output 45
4.4 Convert Existing Backup Directory to Single Image 45
4.5 Extract Existing Image to Backup Directory 45
4.6 List Single-File Backup Contents 45
4.7 Validate a Single-File Backup 45
4.8 Extract Single-File Backup into Current Directory 45
4.9 Extract Single-File Backup into a Backup Directory 45
4.10 Selective Extract of Single File 45
4.11 Selective Extract of Single Directory 46
4.12 Dealing with Absolute Path Names 46
4.13 Single-File Backup to a Remote Host 46
4.14 Single-file Backup to a Remote MySQL Server 46
4.15 Stream a Backup Directory to a Remote MySQL Server 47
4.16 Creating a Cloud Backup on Oracle Cloud Infrastructure Object Storage 47
4.17 Creating a Cloud Incremental Backup on Oracle Cloud Infrastructure 48
4.18 Creating a Cloud Backup on an OpenStack Object Storage 48
4.19 Creating a Cloud Backup in an Amazon S3 Bucket 48
4.20 Creating a Cloud Incremental Backup in an Amazon S3 Bucket 48
4.21 Creating a Cloud Backup on GCP Storage Service 48
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 `optimistic-time=YYMMDDHHMMSS` 61
4.25 Optimistic Backup Using the Option `optimistic-time=now` 61
4.26 Optimistic Backup Using the `optimistic-busy-tables` Option 61
4.27 Optimistic and Partial Backup Using both the `optimistic-busy-tables` and `optimistic-time` Options 62
5.1 Restoring a Database Server 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 Backup Directory using `copy-back-and-apply-log` 70
5.11 Applying the Log to a Backup 70
5.12 Restoring a Single-file Backup from an Oracle Cloud Infrastructure (OCI) Object Storage to a MySQL Server 71
5.13 Restoring a Cloud Incremental Backup from an Oracle Cloud Infrastructure (OCI) Object Storage to a MySQL Server 71
5.14 Restoring a Single-file Backup from an OpenStack Object Storage to a MySQL Server 71
5.15 Restoring a Single-file Backup from Amazon S3 to a MySQL Server 71
5.16 Restoring a Single-file Backup from GCP Storage Service to a MySQL Server 71
11.1 Sample `mysqlbackup` Commands Using MySQL Enterprise Backup with Oracle Secure Backup 93
15.1 Backup Using Simple LDAP Authentication 105
15.2 TTS Restore Using Simple LDAP Authentication 105
15.3 Backup Using SALS-based LDAP Authentication 105

15.4 TTS Restore Using SALS-based Authentication 105
18.1 Apply Log to Full Backup 121
20.1 Sample `backup-my.cnf` file 180

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 26.10 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, 2026, Oracle and/or its affiliates.

License Restrictions

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.

Warranty Disclaimer

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.

Restricted Rights Notice

If this is software, software documentation, data (as defined in the Federal Acquisition Regulation), 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," "commercial computer software documentation," or "limited rights data" 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.

Hazardous Applications Notice

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.

Trademark Notice

Oracle, Java, MySQL, and NetSuite 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.

Third-Party Content, Products, and Services Disclaimer

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.

Use of This Documentation

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
http://www.oracle.com/pls/topic/lookup?ctx=acc&id=docacc.

Access to Oracle Support for Accessibility

Oracle customers that have purchased support have access to electronic support through My Oracle Support. For information, visit
http://www.oracle.com/pls/topic/lookup?ctx=acc&id=info or visit `http://www.oracle.com/pls/topic/lookup?ctx=acc&id=trs` if you are hearing impaired.

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 26.10? 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 26.10.0 is a backup utility for MySQL 26.10.0. 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 5.1.7, “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 26.10 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
`ibdata*`	The InnoDB system tablespace, containing multiple InnoDB tables and associated indexes.	Because the original files might change while the backup is in progress, the `apply-log` step applies the same changes to the corresponding backup files.
`*.ibd`	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.

File Name, Pattern, or Extension	Relation to Original Data Files	Notes
`*.ibz`	Compressed form of InnoDB data files from the MySQL data directory.	Produced instead of `.ibd` files in a compressed backup. The `ibdata*` files representing the InnoDB system tablespace also receive this extension in a compressed backup. The `.ibz` files are uncompressed during the `apply-log`, `copy-back`, or `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.
`*.ARM`	ARCHIVE storage engine table metadata.	The database is put into a read-only state while these files are copied. These files are copied unmodified.
`*.ARZ`	ARCHIVE storage engine table data.	The database is put into a read-only state while these files are copied. These files are copied unmodified.

File Name, Pattern, or Extension	Relation to Original Data Files	Notes
`backup-my.cnf`	Records the configuration parameters that specify the layout of the MySQL data files.	Used in restore operations to reproduce the same layout as when the backup was taken.
`ibbackup_ibd_files`	Records names of the `.ibd` files and their space IDs during an incremental backup.	This file is created during an incremental backup. During a restore, the information in the file is used to delete the tables from the full backup that has been removed between the time of the full backup and the time of the incremental backup.
`ibbackup_logfile`	A condensed version of the `ib_logfile*` files from the MySQL data directory.	The InnoDB log files (`ib_logfile*`) are fixed-size files that are continuously updated during the database's operation. For backup purposes, only the changes that are committed while the backup is in progress are needed. These changes are recorded in `ibbackup_logfile`, and used to re-create the `ib_logfile*` files during the apply-log phase.
`ibbackup_redo_log_only`	Created instead of the `ibbackup_logfile` for incremental backups taken with the `--incremental-with-redo-log-only` option.	
`ib_logfile*`	Created in the backup directory by `mysqlbackup` during the `apply-log` phase after the initial backup.	These files are not copied from the original data directory, but rather re-created in the backup directory during the `apply-log` phase after the initial backup, using the changes recorded in the `ibbackup_logfile` file.
Timestamped directory, such as `2011-05-26_13-42-02`	Created by the `--with-timestamp` option. All the backup files go inside this subdirectory.	Use the `--with-timestamp` option to easily keep more than one set of backup data under the same main backup directory.
`datadir` directory	A subdirectory that stores the data files and database subdirectories from the original MySQL instance.	Created under the backup directory by `mysqlbackup`.
binary log files	Binary log files from the server, which are included in a backup by default (except when the backup is created with the `--use-tts` option). They allow a snapshot of the server to be taken, so a server can be cloned to its exact	Saved under the `datadir` 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

File Name, Pattern, or Extension	Relation to Original Data Files	Notes
	state. Using a full backup as a basis, the binary log files that are included with an incremental backup can be used for a point-in-time recovery (PITR), which restores a database to its state at a certain point in time after the last full backup. See Section 5.3, “Point-in-Time Recovery” for details.	files' locations in the backup, is included into the backup as well also under the `datadir` directory. Use the `--skip-binlog` option to exclude the binary log from the backup. 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 `--log-bin` option to specify a different target location for the binary log. Use the `--skip-binlog` option to skip the restoring of the binary log. The binary log files are compressed and saved with the `.bz` extension when being included in a compressed backup. **Notes** • No binary log files are copied into the incremental backup if the `--use-tts` option or the `--start-lsn` option is used. To include binary log files for the period covered by the incremental backup, do not use the `--use-`

File Name, Pattern, or Extension	Relation to Original Data Files	Notes
		`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. • 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 `--use-tts` option). Their inclusion saves the time and resources required for	Saved under the `datadir` 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

File Name, Pattern, or Extension	Relation to Original Data Files	Notes
	fetching the relay logs from the source when the replica is being restored.	point to the files' locations in the backup directory, is included into the backup as well, under the `datadir` directory. Use the `--skip-relaylog` option to exclude the relay log from the backup. 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 `--relay-log` option to specify a different target location for the relay log. Use the `--skip-relaylog` option to skip the restoring of the relay log. No relay log files are restored onto a server with a partial restore. The relay log files are compressed and saved with the `.bz` extension when being included in a compressed backup.
`*.bz`	Compressed binary log or relay log files.	The binary log and relay log files are compressed and saved with the `.bz` extension when being included in a compressed backup. They are decompressed during a restore.
undo log files	Undo log files from the server. See Undo Tablespaces for details. Both active and inactive undo tablespaces are included in the backup. Also, when the `--incremental-with-redo-log-only` option is used for creating incremental backups, `mysqlbackup` creates from the redo log an undo log for the period covered by the incremental backup, and includes it in the backup.	Saved by default under the `datadir` directory inside the backup. Use the `--backup_innodb_undo_directory` option to specify another location for the undo log in the backup. 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; change their restore locations by editing the tablespace_tracker file.

File Name, Pattern, or Extension	Relation to Original Data Files	Notes
		No undo log files are restored onto a server with a partial restore.
`*.uz`	Compressed undo log files.	The undo log files are compressed and saved with the `.uz` extension when being included in a compressed backup. They are decompressed during a restore.
encrypted keyring data file	For a server using the `component_keyring_encrypted_file` keyring component, the keyring data file specified in the `component_keyring_encrypted_file` file on the server is copied over into the backup under the `meta` folder and named `keyring_kef`. For a server using a keyring plugin or component other than `component_keyring_encrypted_file`, a keyring data file is created and saved under the `meta` folder with the name `keyring_kef`.	An encrypted file containing the master key for InnoDB table encryption . See Chapter 6, *Working with Encrypted InnoDB Tablespaces* for detail.
replica status log files	Usually named `master.info` and `relay-log.info`, they are included by default in a backup of a replica database in a replication setup. See Replication Metadata Repositories, for details.	Saved under the `datadir` directory under the backup directory. The copying of these files are skipped during a backup or a restore when the `--skip-relay-log` option is used.
Backup image file	A single-file backup produced by the `backup-to-image` option, with a name specified by the `--backup-image` option.	You can move the image file without losing or damaging the contents inside it, then unpack it with `mysqlbackup` using the `extract` command and specifying the same image name with the `--backup-image` option. Although some extra files such as `backup-my.cnf` and the `meta` subdirectory are present in the backup directory, these files are also included in the image file and do not need to be moved along with it.
Any other files in subdirectories under the `datadir` directory (that is, under *`backup-dir/datadir/subdir`*)	Copied from the database subdirectories under the MySQL data directory.	By default, any unrecognized files in subdirectories under the MySQL data directory are copied to the backup. To omit such files, specify the `--only-known-file-types` option.

File Name, Pattern, or Extension	Relation to Original Data Files	Notes
		Notes • Some limitations apply to this behavior. See the discussion here in Appendix B, *Limitations of MySQL Enterprise Backup*. • Any subdirectories under a subdirectory under `datadir` (for example, *`backup-dir/datadir/subdir/subdir`*) are ignored in the back up process.
`meta` directory	A subdirectory that stores files with metadata about the backup.	Created under the backup directory by `mysqlbackup`. All files listed below go inside the `meta` subdirectory.
`backup_variables.txt`	Holds important information about the backup. For use by `mysqlbackup` only.	`mysqlbackup` consults and possibly updates this file during operations after the initial backup, such as the apply-log phase or the restore phase.
`image_files.xml`	Contains the list of all the files (except itself) that are present in the single-file backup produced by the `backup-to-image` or `backup-dir-to-image` options. For details about this file, see Section 16.4, “Using the MySQL Enterprise Backup Manifest”.	This file is not modified at any stage once generated.

File Name, Pattern, or Extension	Relation to Original Data Files	Notes
`backup_create.xml`	Lists the command line arguments and environment in which the backup was created. For details about this file, see Section 16.4, "Using the MySQL Enterprise Backup Manifest".	This file is not modified once it is created. You can prevent this file from being generated by specifying the `--disable-manifest` option.
`backup_content.xml`	Essential metadata for the files and database definitions of the backup data. It also contains details of all the plugins defined on the backed-up server, by which users should make sure the same plugins are defined in the same manner on the target server for restoration. For details about this file, see Section 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 `--disable-manifest` option.
`comments.txt`	Produced by the `--comments` or `--comments-file` option.	The comments are specified by you to document the purpose or special considerations for this backup job.
`backup_gtid_executed.sql`	Signifies the backup came from a server with GTIDs enabled.	GTIDs are a replication feature in MySQL 5.6 and higher. See Replication with Global Transaction Identifiers for details. When you back up a server with GTIDs enabled using `mysqlbackup`, the file named `backup_gtid_executed.sql` is created in the `meta` folder under the backup directory. Edit and execute this file after restoring the backup data on a replica server; see Section 8.1, "Setting Up a New replica" for details. **Note** For TTS backups for replica servers, use the `--replica-info` option to have `backup_gtid_execut` generated.

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

File Name, Pattern, or Extension	Relation to Original Data Files	Notes
		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. **Warning** When using the file to restart the target server, change parameters like `--tmpdir`, `--general-log`, etc., and any global variable that uses an absolute file path to avoid the accidental usage of the wrong file locations by the target server.
`backup-auto.cnf`	Copy of the file `auto.cnf` 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 `auto.cnf` before you start the server.
`backup-mysqld-auto.cnf`	Copy of the file `mysqld-.cnf` 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

File Name, Pattern, or Extension	Relation to Original Data Files	Notes
		server, rename the file back to `mysqld-auto.cnf` before you start the server.
`ib_buffer_pool`	The file produced on the server when `innodb_buffer_pool_dump_at_shutdown` (enabled by default on MySQL 5.7.7 and after) or `innodb_buffer_pool_dump_now` is enabled. It holds the list of tablespace IDs and page IDs of the server's buffer pool. The actual file name might be different, as it can be configured by the server's system variable `innodb_buffer_pool_filename`"	With the default setting on MySQL server 5.7.7 and after (`innodb_buffer_pool_load_at_startup=` the target server, during start up, 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.
`tablespace_tracker`	The file tracks external tablespaces, recording their file paths on the backed-up server and their tablespace IDs.	If any external tablespace exists on a backed up server, the tracker file is going to be found in the `datadir` folder inside the backup. Change `server_file_path` 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 `extract` command. **Warnings** • If the tracker file is deleted from the backup, the restore of the backup might fail silently, resulting in corruptions of the restored data.

File Name, Pattern, or Extension	Relation to Original Data Files	Notes
		• You cannot change the restore location for the default undo tablespaces of the database by editing their `server_file_path` entries. Their restore location is controlled by the setting of the `mysqlbackup` option `--innodb_undo_direc` 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.

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 backed-up files, as they are originally copied, form a raw backup that requires further processing. 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) updates the backed-up files based on the changes recorded in the `ibbackup_logfile` file. At this point, the data corresponds to a single point in time.

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, 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 are 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”).

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 (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 `/usr/bin/mysqlbackup`.

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 26.10 > 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 26.10?

MySQL Enterprise Backup 26.10 is an Innovation release, which means it will have new features in addition to bug fixes, and it is supported until the next Innovation release comes out. MySQL Enterprise Backup 26.10 is recommended for use on production systems. With this new Innovation series, the existing 8.4 series will focus on security and bug fixes only.

For notes detailling the changes in MySQL Enterprise Backup 26.10, see MySQL Enterprise Backup 26.10 Release Notes

Part II Using MySQL Enterprise Backup

Table of Contents

4 Backing Up a Database Server ... 31
4.1 Before the First Backup ... 31
4.1.1 Collect Database Information ... 31
4.1.2 Grant MySQL Privileges to Backup Administrator ... 32
4.1.3 Designate a Location for the Backup Directory ... 36
4.2 The Typical Backup / Verify / Restore Cycle ... 36
4.2.1 OS User for Running mysqlbackup ... 36
4.2.2 Backing Up an Entire MySQL Instance ... 37
4.2.3 Verifying a Backup ... 40
4.2.4 Restoring a Database ... 41
4.3 Backup Scenarios and Examples ... 44
4.3.1 Making a Single-File Backup ... 44
4.3.2 Making a Full Backup ... 49
4.3.3 Making a Differential or Incremental Backup ... 50
4.3.4 Making a Compressed Backup ... 56
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 Server ... 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 `--use-tts` Option ... 69
5.1.6 Restoring External InnoDB Tablespaces to Different Locations ... 70
5.1.7 Advanced: Preparing and Restoring a Directory Backup ... 70
5.2 Restoring a Backup from Cloud Storage to a MySQL Server ... 70
5.3 Point-in-Time Recovery ... 71
5.4 Restoring a Backup with a Database Server 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 ... 84
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 Using MySQL Enterprise Backup with MySQL Enterprise Firewall ... 103
15 Using LDAP for Server Authentication ... 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 111

Chapter 4 Backing Up a Database Server

Table of Contents

4.1 Before the First Backup 31
4.1.1 Collect Database Information 31
4.1.2 Grant MySQL Privileges to Backup Administrator 32
4.1.3 Designate a Location for the Backup Directory 36
4.2 The Typical Backup / Verify / Restore Cycle 36
4.2.1 OS User for Running mysqlbackup 36
4.2.2 Backing Up an Entire MySQL Instance 37
4.2.3 Verifying a Backup 40
4.2.4 Restoring a Database 41
4.3 Backup Scenarios and Examples 44
4.3.1 Making a Single-File Backup 44
4.3.2 Making a Full Backup 49
4.3.3 Making a Differential or Incremental Backup 50
4.3.4 Making a Compressed Backup 56
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 `--defaults-file` option in the `mysqld` startup script.	The preferred way to convey database configuration information to `mysqlbackup` is to use the `--defaults-file` option. When connection and data layout information is available from the

Information to Gather	Where to Find It	How to Use It
		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 `--port` option of `mysqlbackup`. The specification is not needed if the information is available from the MySQL configuration file.
Path to MySQL data directory	MySQL configuration file or `mysqld` startup script.	Used to retrieve files from the database instance during backup operations, and to copy files back to the database instance during restore operations. Automatically retrieved from database connection.
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 `--password` option of the `mysqlbackup`. Prompted at the terminal if the `--password` option is present without the password argument.
Path under which to store backup data 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 `mysqlbackup` 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 perform the backup and restore using a different OS user than the one who runs the server, this information might become important. See Section 4.2.1, "OS User for Running mysqlbackup" for details.
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 in Creating Incremental Backups Using Only the Redo Log.	Only needed if you perform incremental backups using the `--incremental-with-redo-log-only` option rather than the `--incremental` option. The size of the InnoDB redo log and the rate of generation for redo data dictate how often you must perform incremental backups.

4.1.2 Grant MySQL Privileges to Backup Administrator

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 limited 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:

- `SELECT` on all databases and tables, for table locks that protect the backups against inconsistency caused by parallel DDL operations.
- `BACKUP_ADMIN` on all databases and tables. This also optimizes locking in order to minimize disruption to database processing.
- `RELOAD` on all databases and tables.
- `SYSTEM_VARIABLES_ADMIN`, to enable and disable logging.
- `REPLICATION CLIENT`, to retrieve the binary log position, which is stored with the backup.
- `PROCESS`, to process DDL statements with the `ALGORITHM = INPLACE` clause.
- `CREATE`, `INSERT`, `DROP`, and `UPDATE` on the tables `mysql.backup_progress` and `mysql.backup_history`, and also `SELECT` and `ALTER` on `mysql.backup_history`.

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, BACKUP_ADMIN, RELOAD, PROCESS, SYSTEM_VARIABLES_ADMIN, REPLICATION CLIENT 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';
```

- 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` 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 backing up and restoring user-created non-InnoDB tables:
    - `LOCK TABLES` on all schemas containing user-created non-InnoDB tables
  - For using redo log archiving for backups:
    - `INNODB_REDO_LOG_ARCHIVE` to invoke the `innodb_redo_log_archive_start()` function .
  - For Section 5.1.4, “Table-Level Recovery (TLR)” of non-TTS backups:
    - `INSERT` and `ALTER` to update tables

  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, CREATE, DROP, FILE, INSERT, ALTER ON *.* TO 'mysqlbackup'@'localhost';
  GRANT CREATE, DROP, UPDATE ON mysql.backup_sbt_history TO 'mysqlbackup'@'localhost';
  GRANT ENCRYPTION_KEY_ADMIN ON *.* TO 'mysqlbackup'@'localhost';
  GRANT INNODB_REDO_LOG_ARCHIVE ON *.* TO 'mysqlbackup'@'localhost';
  ```

- For privileges required for using MySQL Enterprise Backup with a Group Replication setting, see Chapter 9, *Using MySQL Enterprise Backup with Group Replication*.
- The following additional privileges might also be required after a server upgrade:
  - **When using MySQL Enterprise Backup 26.10 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 26.10 has taken place on the server, by which point they can be revoked.

- **When using MySQL Enterprise Backup 26.10 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 26.10 has taken place on the server, by which point they can be revoked.

- **When performing for the first time a backup using the SBT API with MySQL Enterprise Backup 26.10 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 26.10 using the SBT API has taken place on the server, by which point they can be revoked.

Note

Make sure that the limit `MAX_QUERIES_PER_HOUR` is not set for the user `mysqlbackup` uses to access the server, or backup operations might fail unexpectedly.

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 OS User for Running mysqlbackup

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

Note

For Linux and other Unix-like platforms, the `mysql` user was typically created using the `-r` and `-s /bin/false` options of the `useradd` command, so that it does not have login permissions to your server host (see Creating the mysql User and Group for details). To switch to the `mysql` user on your OS, use the `--shell=/bin/bash` option for the `su` command:

```
su - mysql --shell=/bin/bash
```

If it is not possible run `mysqlbackup` with the OS user `mysql`, pay attention to the following guidelines:

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

4.2.2 Backing Up an Entire MySQL Instance

In the following illustration, 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-dir=/home/meb/mysql/backup-temp \
   --backup-image=/home/meb/mysql/backups/testback3.mbi --with-timestamp backup-to-image
MySQL Enterprise Backup  Ver 8.4.0-commercial for Linux on x86_64 (MySQL Enterprise - Commercial)
Copyright (c) 2003, 2024, Oracle and/or its affiliates.

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-dir=/home/meb/mysql/backup-temp
--backup-image=/home/meb/mysql/backups/testback3.mbi
--with-timestamp
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!".

Enter password:
240426 13:01:23 MAIN     INFO: Establishing connection to server.
240426 13:01:23 MAIN     INFO: No SSL options specified.
240426 13:01:23 MAIN     INFO: MySQL server version is '8.4.0-commercial'
240426 13:01:23 MAIN     INFO: MySQL server compile os version is 'Linux'
240426 13:01:23 MAIN     INFO: SSL/TLS version used for connection is TLSv1.3
240426 13:01:23 MAIN     INFO: Got some server configuration information from running server.

240426 13:01:23 MAIN     INFO: Establishing connection to server for locking.
240426 13:01:23 MAIN     INFO: No SSL options specified.
240426 13:01:23 MAIN     INFO: Backup directory created: '/home/meb/mysql/backup-temp/2024-04-26_13-01-23'
```

```
240426 13:01:23 MAIN     INFO: MySQL server version_comment is 'MySQL Enterprise Server - Commercial'
240426 13:01:23 MAIN     INFO: Mysqlbackup component not installed.
240426 13:01:23 MAIN     INFO: MEB logfile created at /home/meb/mysql/backup-temp/2024-04-26_13-01-23/meta/MEB

240426 13:01:23 MAIN     INFO: The MySQL server has no active keyring.
-------------------------------------------------------------------
                       Server Repository Options:
-------------------------------------------------------------------
  datadir                         = /home/admin/mysql/mysql-datadir/
  innodb_data_home_dir            =
  innodb_data_file_path           = ibdata1:12M:autoextend
  innodb_log_group_home_dir       = /home/admin/mysql/mysql-datadir/
  innodb_undo_directory           = /home/admin/mysql/mysql-datadir/
  innodb_undo_tablespaces         = 2
  innodb_buffer_pool_filename     = ib_buffer_pool
  innodb_page_size                = 16384
  innodb_checksum_algorithm       = crc32

-------------------------------------------------------------------
                       Backup Config Options:
-------------------------------------------------------------------
  datadir                         = /home/meb/mysql/backup-temp/2024-04-26_13-01-23/datadir
  innodb_data_home_dir            = /home/meb/mysql/backup-temp/2024-04-26_13-01-23/datadir
  innodb_data_file_path           = ibdata1:12M:autoextend
  innodb_log_group_home_dir       = /home/meb/mysql/backup-temp/2024-04-26_13-01-23/datadir
  innodb_undo_directory           = /home/meb/mysql/backup-temp/2024-04-26_13-01-23/datadir
  innodb_undo_tablespaces         = 2
  innodb_buffer_pool_filename     = ib_buffer_pool
  innodb_page_size                = 16384
  innodb_checksum_algorithm       = crc32

Backup Image Path = /home/meb/mysql/backups/testback3.mbi
240426 13:01:23 MAIN     INFO: Unique generated backup id for this is 17141508838810794

240426 13:01:23 MAIN     INFO: Copying the server config file '/home/admin/mysql/mysql-datadir/auto.cnf'
240426 13:01:23 MAIN     INFO: Creating 14 buffers each of size 16777216.
240426 13:01:23 MAIN     INFO: The server is not configured for redo log archiving. The system variable innodb
240426 13:01:23 MAIN     INFO: Found checkpoint at lsn 20525419.
240426 13:01:23 MAIN     INFO: Starting log scan from lsn = 20525056 at offset = 870763 and checkpoint = 20525
240426 13:01:23 MAIN     INFO: Full Image Backup operation starts with following threads
                               1 read-threads    6 process-threads    1 write-threads
240426 13:01:23 RDR1     INFO: Copying meta file /home/meb/mysql/backup-temp/2024-04-26_13-01-23/backup-my.cnf
240426 13:01:23 RDR1     INFO: Copying meta file /home/meb/mysql/backup-temp/2024-04-26_13-01-23/meta/backup_c
240426 13:01:23 RDR1     INFO: Copying meta file /home/meb/mysql/backup-temp/2024-04-26_13-01-23/datadir/backu
240426 13:01:23 RLP1     INFO: Starting to parse redo log at lsn = 20525297, whereas checkpoint_lsn = 20525419
240426 13:01:23 RDR1     INFO: Starting to copy all innodb files...
240426 13:01:23 RDR1     INFO: Copying /home/admin/mysql/mysql-datadir/ibdata1.
240426 13:01:24 RDR1     INFO: Starting to copy all undo files...
240426 13:01:24 RDR1     INFO: Copying /home/admin/mysql/mysql-datadir/undo_002.
240426 13:01:24 RDR1     INFO: Copying /home/admin/mysql/mysql-datadir/undo_001.
240426 13:01:24 RDR1     INFO: Starting to lock instance for backup...
240426 13:01:24 RDR1     INFO: The server instance is locked for backup.
240426 13:01:24 RDR1     INFO: The server instance is unlocked after 0.003 seconds.
240426 13:01:24 RDR1     INFO: Copying /home/admin/mysql/mysql-datadir/sys/sys_config.ibd.
240426 13:01:24 RDR1     INFO: Copying /home/admin/mysql/mysql-datadir/pets/cats.ibd.
240426 13:01:24 RDR1     INFO: Copying /home/admin/mysql/mysql-datadir/mysql/backup_progress.ibd.
240426 13:01:24 RDR1     INFO: Copying /home/admin/mysql/mysql-datadir/mysql/backup_history.ibd.
240426 13:01:24 RDR1     INFO: Copying /home/admin/mysql/mysql-datadir/mysql.ibd.
240426 13:01:24 RDR1     INFO: Completing the copy of innodb files.
240426 13:01:24 RDR1     INFO: Requesting a dump of the InnoDB buffer pool
240426 13:01:24 RDR1     INFO: Waiting for the dump of the InnoDB buffer pool to complete
240426 13:01:24 RDR1     INFO: The dump of the InnoDB buffer pool completed
240426 13:01:24 RDR1     INFO: Binary Log Basename: '/home/admin/mysql/mysql-datadir/binlog'
240426 13:01:24 RDR1     INFO: Binary Log Index:    '/home/admin/mysql/mysql-datadir/binlog.index'
240426 13:01:24 RDR1     INFO: Relay Channel:      'group_replication_applier'
240426 13:01:24 RDR1     INFO: Relay Log Basename: '/home/admin/mysql/mysql-datadir/admin-XBox3-relay-bin-grou
240426 13:01:24 RDR1     INFO: Relay Channel:      'group_replication_recovery'
```

```
240426 13:01:24 RDR1     INFO: Relay Log Basename: '/home/admin/mysql/mysql-datadir/admin-XBox3-relay-bin-g
240426 13:01:24 RDR1     INFO: Starting to copy Binlog files.
240426 13:01:24 RDR1     INFO: Starting to lock instance for backup...
240426 13:01:24 RDR1     INFO: The server instance is locked for backup.
240426 13:01:24 RDR1     INFO: The MySQL server has no active keyring.
240426 13:01:24 RDR1     INFO: Requesting flush of redo log reading after LSN 20530626.
240426 13:01:24 RDR1     INFO: Requesting flush of redo log processing after LSN 20530626.
240426 13:01:24 RDR1     INFO: Completed flush of redo log reading after LSN 20531088.
240426 13:01:24 RDR1     INFO: Completed flush of redo log processing after LSN 20531088.
240426 13:01:24 RDR1     INFO: Starting to read-lock tables...
240426 13:01:24 RDR1     INFO: No tables to read-lock.
240426 13:01:24 RDR1     INFO: Opening backup source directory '/home/admin/mysql/mysql-datadir'
240426 13:01:24 RDR1     INFO: Starting to copy non-innodb files in subdirs of '/home/admin/mysql/mysql-da
240426 13:01:24 WTR1     INFO: Adding database directory: datadir/mysql
240426 13:01:24 WTR1     INFO: Adding database directory: datadir/performance_schema
240426 13:01:24 RDR1     INFO: Completing the copy of all non-innodb files.
240426 13:01:24 WTR1     INFO: Adding database directory: datadir/pets
240426 13:01:24 WTR1     INFO: Adding database directory: datadir/sys
240426 13:01:24 RDR1     INFO: Requesting consistency information...
240426 13:01:24 RDR1     INFO: Locked the consistency point for 1902 microseconds.
240426 13:01:24 RDR1     INFO: Consistency point server_uuid 'c6d42eaa-03e9-11ef-ae5e-0800276c22b0'.
240426 13:01:24 RDR1     INFO: Consistency point gtid_executed ''.
240426 13:01:24 RDR1     INFO: Consistency point binary_log_file 'binlog.000001'.
240426 13:01:24 RDR1     INFO: Consistency point binary_log_position 3807.
240426 13:01:24 RDR1     INFO: Consistency point InnoDB lsn 20531571.
240426 13:01:24 RDR1     INFO: Consistency point InnoDB lsn_checkpoint 20525419.
240426 13:01:24 RDR1     INFO: Requesting completion of redo log copy after LSN 20531571.
240426 13:01:24 RLW1     INFO: A copied database page was modified at 20529721. (This is the highest lsn fo
240426 13:01:24 RLW1     INFO: Scanned log up to lsn 20531571.
240426 13:01:24 RLW1     INFO: Was able to parse the log up to lsn 20531571.
240426 13:01:24 RLW1     INFO: Copied redo log
                               log_start_lsn    20525056
                               start_checkpoint 20525419
                               start_lsn        20525419
                               last_checkpoint  20525419
                               consistency_lsn  20531571
                               log_end_lsn      20531571
240426 13:01:24 RLR1     INFO: Redo log reader waited 87 times for a total of 435.00 ms for logs to genera
240426 13:01:24 RDR1     INFO: Truncating binary log index '/home/meb/mysql/backup-temp/2024-04-26_13-01-23
240426 13:01:24 RDR1     INFO: Truncating binary log 'binlog.000001' to 3807.
240426 13:01:24 RDR1     INFO: Copying /home/admin/mysql/mysql-datadir/binlog.000001.
240426 13:01:24 RDR1     INFO: Completed the copy of binlog files...
240426 13:01:24 RDR1     INFO: The server instance is unlocked after 0.130 seconds.
240426 13:01:24 RDR1     INFO: Reading all global variables from the server.
240426 13:01:24 RDR1     INFO: Completed reading of all 623 global variables from the server.
240426 13:01:24 RDR1     INFO: Writing server defaults files 'server-my.cnf' and 'server-all.cnf' for serve
240426 13:01:24 RDR1     INFO: Copying meta file /home/meb/mysql/backup-temp/2024-04-26_13-01-23/meta/backu
240426 13:01:24 RDR1     INFO: Copying meta file /home/meb/mysql/backup-temp/2024-04-26_13-01-23/datadir/ib
240426 13:01:24 RDR1     INFO: Copying meta file /home/meb/mysql/backup-temp/2024-04-26_13-01-23/server-all
240426 13:01:24 RDR1     INFO: Copying meta file /home/meb/mysql/backup-temp/2024-04-26_13-01-23/server-my
240426 13:01:24 RDR1     INFO: Copying meta file /home/meb/mysql/backup-temp/2024-04-26_13-01-23/meta/backu
240426 13:01:24 RDR1     INFO: Copying meta file /home/meb/mysql/backup-temp/2024-04-26_13-01-23/meta/image
240426 13:01:24 MAIN     INFO: Full Image Backup operation completed successfully.
240426 13:01:24 MAIN     INFO: Backup image created successfully.
240426 13:01:24 MAIN     INFO: Image Path = /home/meb/mysql/backups/testback3.mbi
240426 13:01:24 MAIN     INFO: MySQL binlog position: filename binlog.000001, position 3807.

------------------------------------------------------------
   Parameters Summary
------------------------------------------------------------
   Start LSN                  : 20525056
   Last Checkpoint LSN        : 20525419
   End LSN                    : 20531571
------------------------------------------------------------

mysqlbackup completed OK!
```

4.2.3 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/meb/mysql/backups/testback3.mbi validate
MySQL Enterprise Backup  Ver 8.4.0-commercial for Linux on x86_64 (MySQL Enterprise - Commercial)
Copyright (c) 2003, 2024, Oracle and/or its affiliates.

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
--backup-image=/home/meb/mysql/backups/testback3.mbi
validate

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

240426 13:09:04 MAIN     INFO: Backup Image MEB version string: 8.4.0
240426 13:09:04 MAIN     INFO: MySQL server version is '8.4.0'
240426 13:09:04 MAIN     INFO: The backup image has no keyring.
240426 13:09:04 MAIN     INFO: Creating 14 buffers each of size 16777216.
240426 13:09:04 MAIN     INFO: Validate operation starts with following threads
                               1 read-threads    6 process-threads
240426 13:09:04 MAIN     INFO: Validating image ... /home/meb/mysql/backups/testback3.mbi
240426 13:09:04 PCR1     INFO: Validate: [Dir]: meta
240426 13:09:04 PCR3     INFO: Validate: [Dir]: datadir/mysql
240426 13:09:04 PCR3     INFO: Validate: [Dir]: datadir/performance_schema
240426 13:09:04 PCR6     INFO: Validate: [Dir]: datadir/pets
240426 13:09:04 PCR6     INFO: Validate: [Dir]: datadir/sys
240426 13:09:04 MAIN     INFO: datadir/mysql.ibd validated.
240426 13:09:04 MAIN     INFO: datadir/mysql/backup_history.ibd validated.
240426 13:09:04 MAIN     INFO: datadir/mysql/backup_progress.ibd validated.
240426 13:09:04 MAIN     INFO: datadir/sys/sys_config.ibd validated.
240426 13:09:04 MAIN     INFO: datadir/undo_001 validated.
240426 13:09:04 MAIN     INFO: datadir/pets/cats.ibd validated.
240426 13:09:04 MAIN     INFO: datadir/undo_002 validated.
240426 13:09:04 MAIN     INFO: datadir/ibdata1 validated.
240426 13:09:04 MAIN     INFO: Validate operation completed successfully.
240426 13:09:04 MAIN     INFO: Backup Image validation successful.
240426 13:09:04 MAIN     INFO: Source Image Path = /home/meb/mysql/backups/testback3.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 4.2.4, “Restoring a Database” for the basic steps for restoring a backup, and see Chapter 5, *Recovering or Restoring a Database Server* for more detailed instructions.

Warning

Do not try to verify a backup by starting a MySQL Server using a backup directly. **You should never start a server using a raw backup directory as a data directory**, as the two types of directories are different and the action will crash the server and might corrupt your backup. Start the server only after performing the proper restore operation. See Appendix A, *Frequently Asked Questions for MySQL Enterprise Backup* for details."

4.2.4 Restoring a Database

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

- Shut down the database server.
- Delete all files inside the server's data directory. Also delete all files inside the directories specified by the `--innodb_data_home_dir`, `--innodb_log_group_home_dir`, and `--innodb_undo_directory` options for restore, if the directories are different from the data directory.
- Use, for example, the `copy-back-and-apply-log` command, which converts the raw backup into a prepared backup by updating it to a consistent state, and then copies the tables, indexes, metadata, and any other required files onto a target server. For the various options that you can specify for this operation, see Section 18.3, “Restore Operations”.

In the illustration below, the single-file backup created in the example given in Section 4.2.2, “Backing Up an Entire MySQL Instance” is restored using the `copy-back-and-apply-log` command. 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/mysql/mysql-datadir \
  --backup-image=/home/meb/mysql/backups/testback3.mbi --backup-dir=/home/meb/mysql/backup-tmp2 \
    copy-back-and-apply-log
MySQL Enterprise Backup  Ver 8.4.0-commercial for Linux on x86_64 (MySQL Enterprise - Commercial)
Copyright (c) 2003, 2024, Oracle and/or its affiliates.

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/mysql/mysql-datadir
--backup-image=/home/meb/mysql/backups/testback3.mbi
--backup-dir=/home/meb/mysql/backup-tmp2
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!".

240426 13:17:07 MAIN     INFO: Backup Image MEB version string: 8.4.0
240426 13:17:07 MAIN     INFO: MySQL server version is '8.4.0'
240426 13:17:07 MAIN     INFO: Backup directory created: '/home/meb/mysql/backup-tmp2'
240426 13:17:07 MAIN  WARNING: If you restore to a server of a different version, the innodb_data_file_pat
240426 13:17:07 MAIN     INFO: MEB logfile created at /home/meb/mysql/backup-tmp2/meta/MEB_2024-04-26.13-1

240426 13:17:07 MAIN     INFO: The backup image has no keyring.
----------------------------------------------------------------------
                        Server Repository Options:
----------------------------------------------------------------------
  datadir                         = /home/admin/mysql/mysql-datadir
  innodb_data_home_dir            = /home/admin/mysql/mysql-datadir
  innodb_data_file_path           = ibdata1:12M:autoextend
  innodb_log_group_home_dir       = /home/admin/mysql/mysql-datadir
```

```
  innodb_undo_directory          = /home/admin/mysql/mysql-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/meb/mysql/backup-tmp2/datadir
  innodb_data_home_dir           = /home/meb/mysql/backup-tmp2/datadir
  innodb_data_file_path          = ibdata1:12M:autoextend
  innodb_log_group_home_dir      = /home/meb/mysql/backup-tmp2/datadir
  innodb_undo_directory          = /home/meb/mysql/backup-tmp2/datadir
  innodb_undo_tablespaces        = 2
  innodb_buffer_pool_filename    = ib_buffer_pool
  innodb_page_size               = 16384
  innodb_checksum_algorithm      = crc32

240426 13:17:07 MAIN     INFO: Creating 14 buffers each of size 16777216.
240426 13:17:07 MAIN     INFO: Copy-back-and-apply-log from image operation starts with following threads
                               1 read-threads    6 process-threads    1 write-threads
240426 13:17:07 RDR1     INFO: Copying ibdata1.
240426 13:17:07 PCR3     INFO: Copying database directory: meta
240426 13:17:07 RDR1     INFO: Copying undo_002.
240426 13:17:07 RDR1     INFO: Copying undo_001.
240426 13:17:07 RDR1     INFO: Copying sys/sys_config.ibd.
240426 13:17:07 RDR1     INFO: Copying pets/cats.ibd.
240426 13:17:07 RDR1     INFO: Copying mysql/backup_progress.ibd.
240426 13:17:07 RDR1     INFO: Copying mysql/backup_history.ibd.
240426 13:17:07 RDR1     INFO: Copying mysql.ibd.
240426 13:17:07 PCR1     INFO: Copying database directory: mysql
240426 13:17:07 PCR1     INFO: Copying database directory: performance_schema
240426 13:17:08 PCR5     INFO: Copying database directory: pets
240426 13:17:08 PCR5     INFO: Copying database directory: sys
240426 13:17:08 RDR1     INFO: Binary Log Basename: 'binlog'
240426 13:17:08 RDR1     INFO: Binlog Log Index:    '/home/admin/mysql/mysql-datadir/binlog.index'
240426 13:17:08 RDR1     INFO: Copying binlog.000001.
240426 13:17:08 MAIN     INFO: read_backup_variables_txt_file: '/home/meb/mysql/backup-tmp2/meta/backup_variab
240426 13:17:08 MAIN     INFO: backup variable mysql_version=8.4.0-commercial
240426 13:17:08 MAIN     INFO: MySQL server version is '8.4.0-commercial'
240426 13:17:08 MAIN     INFO: Restoring ...8.4.0-commercial version
240426 13:17:08 MAIN     INFO: backup variable meb_version=8.4.0
240426 13:17:08 MAIN     INFO: backup variable start_lsn=20525056
240426 13:17:08 MAIN     INFO: backup variable last_checkpoint=20525419
240426 13:17:08 MAIN     INFO: backup variable end_lsn=20531571
240426 13:17:08 MAIN     INFO: backup variable apply_log_done=0
240426 13:17:08 MAIN     INFO: backup variable is_incremental=0
240426 13:17:08 MAIN     INFO: backup variable is_incremental_with_redo_log_only=0
240426 13:17:08 MAIN     INFO: backup variable is_partial=0
240426 13:17:08 MAIN     INFO: backup variable is_compressed=0
240426 13:17:08 MAIN     INFO: backup variable is_skip_binlog=0
240426 13:17:08 MAIN     INFO: backup variable is_skip_relaylog=0
240426 13:17:08 MAIN     INFO: backup variable is_skip_unused_pages=0
240426 13:17:08 MAIN     INFO: backup variable is_onlyinnodb=0
240426 13:17:08 MAIN     INFO: backup variable binlog_position=binlog.000001:3807
240426 13:17:08 MAIN     INFO: backup variable binlog_index=binlog.index
240426 13:17:08 MAIN     INFO: backup variable has_tde_tables=0
240426 13:17:08 MAIN     INFO: backup variable start_time_utc=1714150880468548
240426 13:17:08 MAIN     INFO: backup variable end_time_utc=1714150884925922
240426 13:17:08 MAIN     INFO: backup variable consistency_time_utc=1714150884898388
240426 13:17:08 MAIN     INFO: backup variable mysql_version_comment=MySQL Enterprise Server - Commercial
240426 13:17:08 MAIN     INFO: backup variable log_bin_name=binlog
240426 13:17:08 MAIN     INFO: backup variable log_bin_index_name=binlog
240426 13:17:08 MAIN     INFO: backup variable innodb_undo_files_count=2
240426 13:17:08 MAIN     INFO: Copy-back operation completed successfully.
240426 13:17:08 MAIN     INFO: Source Image Path = /home/meb/mysql/backups/testback3.mbi
```

```
240426 13:17:08 MAIN     INFO: read_backup_variables_txt_file: '/home/meb/mysql/backup-tmp2/meta/backup_va
240426 13:17:08 MAIN     INFO: backup variable mysql_version=8.4.0-commercial
240426 13:17:08 MAIN     INFO: MySQL server version is '8.4.0-commercial'
240426 13:17:08 MAIN     INFO: Restoring ...8.4.0-commercial version
240426 13:17:08 MAIN     INFO: backup variable meb_version=8.4.0
240426 13:17:08 MAIN     INFO: backup variable start_lsn=20525056
240426 13:17:08 MAIN     INFO: backup variable last_checkpoint=20525419
240426 13:17:08 MAIN     INFO: backup variable end_lsn=20531571
240426 13:17:08 MAIN     INFO: backup variable apply_log_done=0
240426 13:17:08 MAIN     INFO: backup variable is_incremental=0
240426 13:17:08 MAIN     INFO: backup variable is_incremental_with_redo_log_only=0
240426 13:17:08 MAIN     INFO: backup variable is_partial=0
240426 13:17:08 MAIN     INFO: backup variable is_compressed=0
240426 13:17:08 MAIN     INFO: backup variable is_skip_binlog=0
240426 13:17:08 MAIN     INFO: backup variable is_skip_relaylog=0
240426 13:17:08 MAIN     INFO: backup variable is_skip_unused_pages=0
240426 13:17:08 MAIN     INFO: backup variable is_onlyinnodb=0
240426 13:17:08 MAIN     INFO: backup variable binlog_position=binlog.000001:3807
240426 13:17:08 MAIN     INFO: backup variable binlog_index=binlog.index
240426 13:17:08 MAIN     INFO: backup variable has_tde_tables=0
240426 13:17:08 MAIN     INFO: backup variable start_time_utc=1714150880468548
240426 13:17:08 MAIN     INFO: backup variable end_time_utc=1714150884925922
240426 13:17:08 MAIN     INFO: backup variable consistency_time_utc=1714150884898388
240426 13:17:08 MAIN     INFO: backup variable mysql_version_comment=MySQL Enterprise Server - Commercial
240426 13:17:08 MAIN     INFO: backup variable log_bin_name=binlog
240426 13:17:08 MAIN     INFO: backup variable log_bin_index_name=binlog
240426 13:17:08 MAIN     INFO: backup variable innodb_undo_files_count=2
240426 13:17:08 MAIN     INFO: Creating 14 buffers each of size 65536.
240426 13:17:08 MAIN     INFO: Apply-log operation starts with following threads
                               1 read-threads    1 process-threads    6 apply-threads
240426 13:17:08 MAIN     INFO: Using up to 100 MB of memory.
240426 13:17:08 MAIN     INFO: ibbackup_logfile's creation parameters:
          start lsn 20525056, end lsn 20531571,
          start checkpoint 20525419.
240426 13:17:08 MAIN     INFO: Loading the space id : 0, space name : /home/admin/mysql/mysql-datadir/ibda
240426 13:17:08 MAIN     INFO: Loading the space id 4 name '/home/admin/mysql/mysql-datadir/mysql/backup_h
240426 13:17:08 MAIN     INFO: Loading the space id 3 name '/home/admin/mysql/mysql-datadir/mysql/backup_p
240426 13:17:08 MAIN     INFO: Loading the space id 2 name '/home/admin/mysql/mysql-datadir/pets/cats.ibd'
240426 13:17:08 MAIN     INFO: Loading the space id 1 name '/home/admin/mysql/mysql-datadir/sys/sys_config
240426 13:17:08 MAIN     INFO: Loading the space id 4294967294 name '/home/admin/mysql/mysql-datadir/mysql
240426 13:17:08 MAIN     INFO: Loading the space id 4294967279 name '/home/admin/mysql/mysql-datadir/undo_
240426 13:17:08 MAIN     INFO: Loading the space id 4294967278 name '/home/admin/mysql/mysql-datadir/undo_
240426 13:17:08 PCR1     INFO: Starting to parse redo log at lsn = 20525297, whereas checkpoint_lsn = 2052
240426 13:17:08 PCR1     INFO: Doing recovery: scanned up to log sequence number 20531571.
240426 13:17:08 PCR1     INFO: Starting to apply a batch of log records to the database....
InnoDB: Progress in percent: 7 14 21 28 35
240426 13:17:08 PCR1     INFO: Create redo log files. target start_lsn 0 last_checkpoint 0 end_lsn 0
240426 13:17:08 PCR1     INFO: Create redo log files. source start_lsn 20525056 last_checkpoint 20525419 e
240426 13:17:08 PCR1     INFO: Updating last checkpoint to 20525419 in redo log
240426 13:17:08 PCR1     INFO: We were able to parse ibbackup_logfile up to lsn 20531571.
240426 13:17:08 PCR1     INFO: Last MySQL binlog file position 0 3807, file name binlog.000001
240426 13:17:08 PCR1     INFO: The first data file is '/home/admin/mysql/mysql-datadir/ibdata1'
                               and the new created log files are at '/home/admin/mysql/mysql-datadir'
240426 13:17:08 MAIN     INFO: Apply-log operation completed successfully.
240426 13:17:08 MAIN     INFO: Full Backup has been restored successfully.

mysqlbackup completed OK! with 1 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_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 --defaults-file=/home/dbadmin/my.cnf --backup-image=/backups/sales.mbi --backup-dir=/backup-tmp ba
```

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 --defaults-file=/home/dbadmin/my.cnf --backup-image=sales.mbi --backup-dir=/backups backup-to-
```

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 --defaults-file=/home/dbadmin/my.cnf --backup-dir=/backups --backup-image=- backup-to-image > 
```

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`.

MySQL Enterprise Backup supports the following types of cloud storage services:

- Oracle Cloud Infrastructure (OCI) Object Storage
- OpenStack Swift or compatible object storage services
- Amazon Simple Storage Service (S3) or compatible storage service.
- GCP object storage

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

This example creates a cloud backup in an Oracle Cloud Infrastructure (OCI) Object Storage bucket using a Pre-Authenticated Request (PAR) URL.

```
mysqlbackup --defaults-file=/home/dbadmin/my.cnf  --backup-dir=/home/dbadmin/backuptmp \
  --with-timestamp --backup-image=- --cloud-service=OCI --cloud-par-url=<bucket_PAR_URL> --cloud-object=bac
  backup-to-image
```

Example 4.17 Creating a Cloud Incremental Backup on Oracle Cloud Infrastructure

This example creates an incremental cloud backup in an Oracle Cloud Infrastructure (OCI) Object Storage bucket, using a Pre-Authenticated Request (PAR) URL.

```
mysqlbackup --defaults-file=/home/dbadmin/my.cnf --backup-dir=/home/dbadmin/backuptmp --with-timestamp \
  --backup-image=- --cloud-service=OCI --cloud-par-url=<bucket_PAR_URL> --cloud-object=backup-inc.bk  \
  --incremental --incremental-base=history:last_backup backup-to-image
```

Example 4.18 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 \
--defaults-file=/home/dbadmin/my.cnf \
--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/dba/opbackuptmpdir \
--backup-image=- \
backup-to-image
```

Example 4.19 Creating a Cloud Backup in an Amazon S3 Bucket

This example creates a cloud backup in an Amazon S3 bucket.

```
mysqlbackup\
--defaults-file=/home/dbadmin/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/dba/s3backupdir --with-timestamp \
--backup-image=- \
backup-to-image
```

Example 4.20 Creating a Cloud Incremental Backup in an Amazon S3 Bucket

This example creates a cloud incremental backup in an Amazon S3 bucket.

```
mysqlbackup\
--defaults-file=/home/dbadmin/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/dba/s3backupdir --with-timestamp \
--backup-image=- --incremental --incremental-base=history:last_backup \
backup-to-image
```

Example 4.21 Creating a Cloud Backup on GCP Storage Service

This example creates a cloud backup on GCP storage service.

```
mysqlbackup\
--defaults-file=/home/dbadmin/my.cnf \
--cloud-service=GCP \
--cloud-bucket=<bucket name> --cloud-object=<object name> \
--cloud-access-key=<access name> --cloud-secret-key=<secret key> \
--backup-dir=/home/dba/backupdir --with-timestamp \
```

```
--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 (see Appendix B, *Limitations of MySQL Enterprise Backup* for some restrictions).

See Section 5.2, "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 4.2.2, "Backing Up an Entire MySQL Instance".

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

Options on Command Line or in Configuration File?

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

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 the InnoDB tables are determined based on the contents of the `InnoDB` redo log. Since the redo log has a maximum size that you know in advance, depending on the size of your database, the amount of DML activities, and the size of the redo log, it usually requires less I/O to read the changes from the redo log than to scan the InnoDB tablespace files to locate the changed pages.

- A system variable, `innodb_redo_log_capacity`, controls the amount of disk space occupied by redo log files. If the redo log files occupy less space than the value of `innodb_redo_log_capacity`, dirty pages are flushed from the buffer pool to tablespace data files less aggressively, allowing the disk space occupied by the redo log files to grow faster. If the space occupied by the redo log files gets close to the specified value, dirty pages are flushed more aggressively, so that the disk space occupied by redo log files is kept within the specified limit. See Configuring Redo Log Capacity for details.

  With the way the redo log is now maintained, it becomes more likely that when an incremental backup using only the redo log is started, the redo log files storing the changes to the database since the last backup are already processed and are no longer available. To prevent that situation, you should register `mysqlbackup` (the MySQL user who creates backups) with the server as an external consumer of the redo log by the following UDF command, before the creation of any data that is to be included in the redo-log-only incremental backup:

  ```
  DO innodb_redo_log_consumer_register();
  ```

  This prevents InnoDB from removing or recycling redo log files that contain transactions not backed up yet by `mysqlbackup`. After each redo-log-only incremental backup, run the following UDF to advance to a new LSN checkpoint, so that the server may now process the redo log files that are no longer required by `mysqlbackup`:

  ```
  DO innodb_redo_log_consumer_advance($lsn);
  ```

  `$lsn` is the highest LSN value included in a finished incremental backup.

  The steps above assume that the connection session that initiated the `DO innodb_redo_log_consumer_register();` UDF is kept open in between the base backup or the last incremental backup and the latest redo-log-only incremental backup. One way to help ensure that is to have the server spawning on the same machine a special client that connects to the Server by, for example, a Unix socket (if it is a Unix machine) through a connection session that initiates the UDF and is then left open for as long as it is needed. That setup will provide a stable connection session for keeping `mysqlbackup` as a consumer of the redo log.

- 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 redo log capacity, and use this ratio to see how often to take an incremental backup. For example, if you are producing 1GB of redo log data per day, and the combined size of your redo clog capacity (which is specified by `innodb_redo_log_capacity`) 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 capacity of your redo log. 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

`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 26.10 installation, by running this command at a `mysql` client connected to the server:

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

- Start page tracking with the following function:

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

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

```
SELECT mysqlbackup_page_track_get_start_lsn();
```

You can stop page tracking with the following function:

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

Note

The previously mentioned functions 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 specifies `--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. Here are some guidelines for assuring enough memory for the operation:

- The default value of 400 [MB] for the `--limit-memory` option allows `mysqlbackup` to handle about 800GB of changed data. Adjust the value for the option according to your data size.
- The page tracking feature uses the memory buffers configured for `mysqlbackup` for sorting the pages. Determine the number of buffers needed for page sorting by the following steps:
  - Before running the incremental backup, perform the following query on the server to determine the `end_lsn` for the base backup:

    ```
    SELECT end_lsn FROM mysql.backup_history WHERE exit_state = 'SUCCESS'
    AND backup_type != 'TTS' AND server_uuid = @@server_uuid
    ORDER BY end_time DESC, end_lsn DESC LIMIT 0,1;
    ```

  - Run the following query on the server to get the number of changed pages since the base backup was created (retry the query if it returns a negative value):

    ```
    SELECT mysqlbackup_page_track_get_changed_page_count(<the above end_lsn>, 0);
    ```

  - Every changed page needs 8 bytes in the sorting buffer. So, multiply the `changed_page_count` value obtained in the last step by 8 to get the number of bytes needed for the sorting buffer.
  - Each buffer has 16 Megabytes (16777216 bytes). So, divide the number of bytes needed for the sorting buffers calculated in the last step by 16777216

and round the result up to the next integer, to get the number of buffers needed for sorting.

- Make sure the value for the option `--number-of-buffers` is no smaller than the number of required sorting buffers you calculated in the last step. Remember that there could be more changed pages created while you are doing this calculation, so you might want to give `mysqlbackup` a few more extra buffers.

- The default memory limit of 400MB should be able to support up to 25 buffers (up to 18 buffers only for cloud backups); increase the memory limit if you need more buffers than that by changing the value of the `--limit-memory` option.

Page tracking creates a file under the server's datadir for collecting information about changed pages. This file keeps growing until page tracking is stopped. If the server is stopped and restarted, a new page tracking file is created, but the old file persists and continues to grow until page tracking is deactivated explicitly. Using a sequence of SQL statements similar to the following, you can purge any old page-tracking data that you no longer need:

```
SELECT mysqlbackup_page_track_set(false);
SELECT mysqlbackup_page_track_purge_up_to(9223372036854775807);
/* Supply to the loadable function the LSN up to which you want to
purge page tracking data. 9223372036854775807 is the highest possible LSN,
which causes all page tracking files to be purged.*/
SELECT mysqlbackup_page_track_set(true);
```

This can be run, for example, before every full backup.

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 back up.

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 increment 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 . 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.

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 always included in an incremental backup, which means the savings for backup resources are less significant when comparing with the case with InnoDB tables.

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` .
- *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=/home/dbadmin/my.cnf --compress --backup-image=backup.img backup-to-image
```

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

```
mysqlbackup --defaults-file=/home/dbadmin/my.cnf --compress-method=zlib --compress-level=5 backup
```

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

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

Warning

Compression ratio is low for encrypted InnoDB tables. Creating a compressed backup with a database containing many encrypted InnoDB tables is not recommended.

Note

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

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

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.*" backup

# 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).*" backup
```

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.

Notes

- An optimistic backup cannot be performed for an incremental backup or a backup using transportable tablespaces (TTS).
- Do not perform a DDL operation on the server in parallel with an optimistic backup, or the backup will fail.

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=/home/dbadmin/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=/home/dbadmin/my.cnf --optimistic-time=now \
 --backup-image=<image-name> --backup-dir=<temp-dir> 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=/home/dbadmin/my.cnf --optimistic-busy-tables="^mydatabase\.mytables-.*" \
 --backup-image=<image-name> --backup-dir=<temp-dir> 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=/home/dbadmin/my.cnf --optimistic-busy-tables="^mydatabase\.mytables-.*" \
  --optimistic-time=100516  --backup-image=<image-name> --backup-dir=<temp-dir>  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 2016/08/16, 0800 PM

mysqlbackup --defaults-file=/home/admin/my.cnf --optimistic-time=160816200000 \
  --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=/home/dbadmin/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__201702052330.b
  --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__201702062330.b
  --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__201702072330.b
  --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:

  ```
  $> 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-=/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 . 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 Server

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 `--use-tts` Option 69
5.1.6 Restoring External InnoDB Tablespaces to Different Locations 70
5.1.7 Advanced: Preparing and Restoring a Directory Backup 70
5.2 Restoring a Backup from Cloud Storage to a MySQL Server 70
5.3 Point-in-Time Recovery 71
5.4 Restoring a Backup with a Database Server 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 server 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 5.1.7, “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 Server

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

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 | last_full_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

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-log
```

See Section 19.13, “Encryption Options” for more details on backup encryption and decryption.

5.1.3 Restoring an Incremental Backup

Note

The `--incremental` option is not needed when restoring an incremental backup.

There are different ways to use incremental backups to restore a database server 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 Server” 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 (in both case, `mysqlbackup` renamed any binary log (but not relay log) files and their index files that have already been restored onto the server by adding the `.old` extension to their file names).

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)

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 .

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.

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/user/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/user/dbadmin/backups/tts-backup.mbi \
```

```
  --include-tables="^sales\.cars" --rename="cars to autos" copy-back-and-apply-log

# A table can be restored into another database; the target database is created if it is not existing on the s
mysqlbackup --socket=/tmp/restoreserver.sock \
  --backup-dir=/BackupDirTemp --backup-image=/home/user/dbadmin/backups/tts-backup.mbi \
  --include-tables="^sales\.cars" --rename="sales.cars to new_sales.autos" copy-back-and-apply-log
```

5.1.6 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.

5.1.7 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 5.10 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 5.11 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`). For a compressed backup, there is no need to add the `--uncompress` option.

- 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 Restoring a Backup from Cloud Storage to a MySQL Server

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

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

```
mysqlbackup --defaults-file=<my.cnf> --backup-dir=/home/user/dbadmin/backuptmp \
  --datadir=<server_datadir> --with-timestamp --backup-image=-  --cloud-service=OCI \
  --cloud-par-url=<backup_PAR_URL> copy-back-and-apply-log
```

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

```
mysqlbackup --defaults-file=<my.cnf> --backup-dir=/home/user/dbadmin/backuptmp \
  --datadir=<server_datadir> --with-timestamp --backup-image=-  --cloud-service=OCI \
  --cloud-par-url=<incremental-backup_PAR_URL> --incremental copy-back-and-apply-log
```

Example 5.14 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 5.15 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 --with-timestamp \
--datadir=/home/user/dba/datadir \
--backup-image=- \
copy-back-and-apply-log
```

Example 5.16 Restoring a Single-file Backup from GCP Storage Service to a MySQL Server

```
mysqlbackup\
--defaults-file=<my.cnf> \
--cloud-service=GCP \
--cloud-bucket=<bucket name> --cloud-object=<object name> \
--cloud-access-key=<access name> --cloud-secret-key=<secret key> \
--backup-dir=/home/user/dba/backuptmpdir --with-timestamp \
--datadir=/home/user/dba/datadir \
--backup-image=- \
copy-back-and-apply-log
```

5.3 Point-in-Time Recovery

You can restore your database server 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). 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 server to its state at the targeted point in time for recovery, identified as `tR` 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 `tR`. 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 `tR`), and pipe the output to the `mysql` client:

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

```
        binary-log-filename  |   mysql -uadmin -p
```

Notes

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

5.4 Restoring a Backup with a Database Server 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 (see In-Place Upgrade for details and limitaions). Here are a number of things that users should pay attention to when restoring a backup with a database server upgrade or downgrade:

- *Restoring a database server 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 26.10 to 8.0) might cause server crashes or data corruption.
- *Restoring a database server 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.

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 a centralized key management solution, 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.

Notes

Since the `keyring_file` and the `keyring_encrypted_file` plugins have been removed from the MySQL Server since release 8.4.0, they are no longer supported by MySQL Enterprise Backup.

Encrypted InnoDB undo logs and Encrypted InnoDB redo logs are supported by MySQL Enterprise Backup. The encrypted undo and redo tablespaces are handled the same way as the encrypted tablespaces for InnoDB tables.

Backing up a database server 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_aws` plugin is used on it.

When the database server 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 or component the server uses. The following is a typical command for backing up a database server containing encrypted InnoDB tablespaces:

```
$ mysqlbackup --defaults-file=/home/dbadmin/my.cnf --backup-image=/home/admin/backups/my.mbi \
  --backup-dir=/home/admin/backup-tmp --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 or component the server is using.
- If the server is using the `component_keyring_encrypted_file` component, 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 `component_keyring_encrypted_file.cnf` file. `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 and names the file `keyring_kef`; the file is encrypted with the password supplied with the option `--encrypt-password`. The encrypted tablespace files are also copied into the backup.
- If the server uses a keyring plugin or component other than component_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`.

Notes

- Backing up a server that uses a keyring plugin or component other than component_keyring_encrypted_file is only supported for servers that allow socket connections or TCP/IP connections using TLS; it is, therefore, not supported when, for example, the server is running on a Windows platform and only allows shared memory connections.
- 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.
- If the server uses the `keyring_hashicorp` plugin, use the `--encrypt-password` to supply the HashiCorp Vault AppRole authentication secret ID, which was the value of `keyring_hashicorp_secret_id` on the server to be backed up.

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 backup 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 server must be supplied with the `--encrypt-password` option for a restore operation. During a restore, `mysqlbackup` copies the encrypted InnoDB tablespace files onto the server. It also performs the following actions:

- *When the component_keyring_encrypted_file keyring component was used on the backed-up server,* `mysqlbackup` restores the encrypted keyring data file to its proper location on the server, and also creates a `manifest` file and the configuration file `component_keyring_encrypted_file.cnf` (which contains the password used with the `--encrypt-password` option during the restore) on the restored server, so that the server will load `component_keyring_encrypted_file` when it restarts.
- *When the component_keyring_file keyring component was used on the backed-up server,* `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. It also creates a `manifest` file and the configuration file `component_keyring_file.cnf` on the restored server, so that the server will load the `component_keyring_file` component when it restarts.
- *When any keyring plugin was used on the backed-up server,* `mysqlbackup` restores the encrypted keyring data file to its proper location on the server. The restored server has to be started with `component_keyring_encrypted_file`. `mysqlbackup` creates a `manifest` file and the configuration file `component_keyring_encrypted_file.cnf` (which contains the password used with the `--encrypt-password` option during the restore) on the restored server, so that the server will load component_keyring_encrypted_file when it restarts.

Take these additional steps after the restore operation is finished:

- *To use global manifest and configuration file for starting the keyring component:*
  - Copy the `manifest` file from restored data directory to the folder where the `mysqld` binary resides.
  - Copy the configuration file `component_keyring_encrypted_file.cnf` or `component_keyring_file.cnf` (depending on how your backup is being restored; see the discussion above) from the restore data directory to the folder where the component binary resides.
- *To use local manifest and configuration file for starting the keyring component:*
  - Create a new `manifest` file with following contents in the folder where the `mysqld` binary resides

    ```
    { "read_local_manifest": true }
    ```

  - Create a new configuration file `component_keyring_encrypted_file.cnf` or `component_keyring_file.cnf` (depending on how your backup is being restored; see the discussion above) with following contents in the folder where the component binary resides :

    ```
    { "read_local_config": true }
    ```

If you want to use another keyring plugin or component (for example, the backed-up server was using `keyring_aws` and you want the restored server to use it too, or you simply want to switch to a new component or plugin), a keyring migration can be performed.

For Incremental Backups. For a series of incremental backups, if a component other than `component_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 same password used to make the specific full or incremental backup must be provided to restore that backup, and, if any keyring plugin is used, 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.

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 --defaults-file=/home/dbadmin/my.cnf --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 supplied during the backup 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 the 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 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

`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 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 84
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-replica-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 REPLICATION SOURCE TO` SQL statement and the information you have retrieved in the last step to initialize the replica properly:

```
CHANGE REPLICATION SOURCE TO
SOURCE_LOG_FILE='hundin-bin.000006',
SOURCE_LOG_POS=128760128;
```

6. Set the status of any events that were copied from the source to `DISABLE ON REPLICA`. For example:

```
mysql> ALTER EVENT mysql.event DISABLE ON REPLICA;
```

7. Remove the line `skip-replica-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-replica-start` entry in, but then you will always need to use the `START REPLICA` 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-replica-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 REPLICATION SOURCE TO` statement for initializing the replica:

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

Uncomment the command and add any needed connection and authentication parameters to it (for example, `SOURCE_HOST`, `SOURCE_USER`, `SOURCE_PASSWORD`, and `SOURCE_PORT`):

```
# Use the following command if you want to use the GTID handshake protocol:
CHANGE REPLICATION SOURCE TO SOURCE_HOST='127.0.0.1', SOURCE_USER='muser', SOURCE_PASSWORD='mpass', SOURCE_
```

Execute the file with the `mysql` client

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

6. Set the status of any events that were copied from the source to `DISABLE ON REPLICA`. For example:

```
mysql> ALTER EVENT mysql.event DISABLE ON REPLICA;
```

7. Remove the `skip-replica-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-replica-start` entry in, but then you will always need to use the `START REPLICA` 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 `--replica-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 a 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 of a replica backup 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 `Replica_open_temp_tables` is 0.

2. After `Replica_open_temp_tables` is detected to be 0, `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 `Replica_open_temp_tables` is still 0

   - If it is, `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, 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-replica-backup-timeout` option.

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

Even though the issue described above with temporary tables does not exist for a row-based replication (RBR) setup (for which temporary tables are not replicated onto the replica), any replica backups still enter the same loop as described above, because the mode of replication could potentially be switched from

RBR to statement-based replication (SBR) or mixed-mode replication at any time. User who are certain that SBR is not occurring for the replica can set `--safe-replica-backup-timeout=0`, which will prevent `mysqlbackup` from entering the above-mentioned loop.

Note

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

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 REPLICA;
   ```

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 REPLICA;
```

8.4 Working with Encrypted Binary and Relay Logs

MySQL Enterprise Backup 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 Tablespaces* 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 `component_keyring_encrypted_file` keyring component, 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 `component_keyring_encrypted_file.cnf` file. `mysqlbackup` then copies over 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 and names the file `keyring_kef`.

- If the server uses a keyring plugin or component other than `component_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. `mysqlbackup` restores the encrypted keyring data file to its proper location on the server. If the backed up server used the `component_keyring_file` keyring component. `mysqlbackup` creates a `manifest` file and the configuration file `component_keyring_file.cnf` (which contains the password used with the `--encrypt-password` option during the restore) on the restored server, so that the server will load `component_keyring_file` when it restarts. Otherwise, `mysqlbackup` creates a `manifest` file and the configuration file `component_keyring_encrypted_file.cnf` (which contains the password used with the `--encrypt-password` option during the restore) on the restored server, so that the server will load `component_keyring_encrypted_file` when it restarts.

If you want to use another keyring plugin or component (for example, the backed-up server was using `keyring_aws` and you want the restored server to use it too, or you simply want to switch to a new component or plugin), a keyring migration can be performed.

For Incremental Backups. For a series of incremental backups, if a keyring plugin or component other than `component_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.

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.

When working with a Group Replication setup, `mysqlbackup` makes the backup information available to all members of the server group by making sure that the `backup_history`, `backup_sbt_history` (when using the SBT API), and `backup_progress` tables are updated on a primary node for each `mysqlbackup` operation. For the feature to work on any host in a Group Replication setup, make sure the following requirements are met:

- Host names or host addresses contained in the `member_host` column in the `performance_schema.replication_group_members` table must be resolvable by `mysqlbackup`.
- In addition to the `mysqlbackup@localhost` account created on each host as described in Section 4.1.2, “Grant MySQL Privileges to Backup Administrator”, accounts for the user `mysqlbackup` to access *all* hosts in the group have been created on *every* member of the server group. These are sample statements for creating the accounts:

  ```
  CREATE USER 'mysqlbackup'@'host1' IDENTIFIED BY 'password';
  CREATE USER 'mysqlbackup'@'host2' IDENTIFIED BY 'password';
  CREATE USER 'mysqlbackup'@'host3' IDENTIFIED BY 'password';
  ...
  ```

  Or, if the host names fall into a pattern, use a wildcard for the host names (if allowed by your environment's security policy) to perform the same task with a single statement; for example:

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

- The account `mysqlbackup@localhost` *AND* all other accounts for accessing the hosts (i.e., `mysqlbackup@host1`, `mysqlbackup@host2`,`mysqlbackup@host3`, etc, or simply `mysqlbackup@host_` in our example) use *the same password* for server access. This is needed because, depending on how host names are resolved, `mysqlbackup` will likely be using two accounts to finish a backup job on a Group Replication set up (one for the actual backup operation, the other for updating the history and progress tables on a primary host), but only one password can be specified in the backup command. Therefore, the accounts must use the same password.
- All the privileges granted to `mysqlbackup@localhost` as described in Section 4.1.2, “Grant MySQL Privileges to Backup Administrator” are also granted to all other accounts with which `mysqlbackup` connects to the hosts:

  ```
  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';
  [More grant statements for other privileges for all hosts ...]
  ```

  Or, if the host names fall into a pattern, use a wildcard for the host names (if allowed by your environment's security policy) to perform the same task; for example:

  ```
  GRANT SELECT ON performance_schema.replication_group_members TO 'mysqlbackup'@'host_';
  [More grant statements for other privileges...]
  ```

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
6a1d325e6ef0577f3400b7cd624ae574f5186d0da2eeb946895de418297ed75b
```

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

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

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

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

The command options used for encryption and decryption are `--encrypt`, `--decrypt`, `--key`, and `--key-file`. These options can be used with various operations on backup images. See Section 19.13, "Encryption Options" for details.

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

```
mysqlbackup --defaults-file=/home/dbadmin/backup.cnf --backup-image=/backups/image.enc --encrypt \
   --key=23D987F3A047B475C900127148F9E039485798364519287 4A2B3049570C12A34 \
   --backup-dir=/var/tmp/backup  backup-to-image
```

To use a key file for the same task:

```
mysqlbackup --defaults-file=/home/dbadmin/backup.cnf --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.

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     | sb
+----+-------------------+------------------+---------------------+---------------------+------------------+---
|  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 26.10 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

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 (not applicable to single-file backups, which always use a single write thread).

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

MyISAM Considerations

Important

- Although `mysqlbackup` backs up InnoDB tables without interrupting database use, the final stage that copies non-InnoDB files (such as MyISAM tables and `.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 server, 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 server. 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 server communication controlled by the `--protocol` option is low-volume. For example, `mysqlbackup` retrieves information about database server parameters through the database server 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 server 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 (except for Table-Level Restore) 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 (not applicable to restores of single-file backups, which always use a single read thread).
- 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 Using MySQL Enterprise Backup with MySQL Enterprise Firewall

MySQL Enterprise Edition includes MySQL Enterprise Firewall, an application-level firewall that enables database server 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 26.10 only uses the statement sporadically, it is easy to miss it in the training phase for the MySQL Enterprise Firewall.

Chapter 15 Using LDAP for Server Authentication

LDAP pluggable authentication is a feature supported by the MySQL Enterprise Edition; the user by which `mysqlbackup` connects to the server can be authenticated using LDAP. Beyond proper setup for LDAP on the server side, the following requirements apply for LDAP authentication to work with `mysqlbackup`:

- If the user is using simple LDAP authentication, the password is communicated to the server in cleartext; enable Client-Side Cleartext Pluggable Authentication by using the `--enable-cleartext-plugin` option in your `mysqlbackup` command, or set the environment variable `LIBMYSQL_ENABLE_CLEARTEXT_PLUGIN=[1|Y|y]` in the environment you run the command.
- If the user is using SASL-based LDAP authentication and the client-side plugin (`authentication_ldap_sasl_client`) is not located in the server's plugin folder, the `--plugin-dir` option should be used with the `mysqlbackup` command to specify the location of the plugin, or the same information should be provided by setting the environment variable `LIBMYSQL_PLUGIN_DIR` in the environment you run the command.

The following are some sample commands for using LDAP authentication:

Example 15.1 Backup Using Simple LDAP Authentication

```
mysqlbackup --user=mysqlbackup --password=password --host=127.0.0.1 \
  --backup-dir=/home/mysqlbackup/backup-tmp  --backup-image=/home/mysqlbackup/backups/my.mbi \
  --enable-cleartext-plugin backup-to-image
```

Example 15.2 TTS Restore Using Simple LDAP Authentication

```
mysqlbackup --user=mysqlbackup --password=password  --host=127.0.0.1 \
  --backup-dir=/home/mysqlbackup/backup-tmp  --backup-image=/home/mysqlbackup/backups/my.mbi \
  --include-tables=REGEX --enable-cleartext-plugin copy-back-and-apply-log
```

Example 15.3 Backup Using SALS-based LDAP Authentication

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

Example 15.4 TTS Restore Using SALS-based Authentication

```
mysqlbackup --user=mysqlbackup --password=password  --host=127.0.0.1 \
  --backup-dir=/home/mysqlbackup/backup-tmp  --backup-image=/home/mysqlbackup/backups/my.mbi \
   --include-tables=REGEX --plugin-dir=SASL_plugin-dir copy-back-and-apply-log
```

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 111

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 exits 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

Exit Code	Exit Message
8	One or more of the arguments are unknown
9	IO operation failed
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 server 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_
| current_state | varchar(200)  | NO   |     | NULL                 |
+---------------+---------------+------+-----+----------------------+-------------------------------------
7 rows in set (0.00 sec)
```

The `backup_progress` table is in InnoDB format.

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                |       |
| 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                |       |
+---------------------------+---------------+------+-----+---------------------+-------+
26 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 the latest backup as the starting LSN value for the 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 server 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 123
18.5 Other Single-File Backup Operations 124
18.6 Other Operations 126
19 mysqlbackup Command-Line Options 129
19.1 General Options 134
19.2 Connection Options 136
19.3 Server Repository Options 137
19.4 Backup Repository Options 140
19.5 Metadata Options 144
19.6 Compression Options 144
19.7 Incremental Backup Options 147
19.8 Partial Backup and Restore Options 149
19.9 Single-File Backup Options 154
19.10 Performance / Scalability / Capacity Options 156
19.11 Message Logging Options 163
19.12 Progress Report Options 164
19.13 Encryption Options 168
19.14 Options for Working with Encrypted InnoDB Tablespaces and Encrypted Binary/Relay Logs 168
19.15 Cloud Storage Options 169
19.16 Options for Special Backup Types 174
20 Configuration Files and Parameters 179

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.
- 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 server 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 123
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-INNODB-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.

  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-INNODB-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-INNODB-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 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 Server*.

```
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`:

  - 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:
    - 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:
    - 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 5.1.7, “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.

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 4.2.3, “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`

  It is an alias for the `extract` command; see the description below for `extract`.

  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.

Use the `--uncompress` option to extract files from a compressed single-file backup (the `--uncompress` option is not required *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 General Options 134
19.2 Connection Options 136
19.3 Server Repository Options 137
19.4 Backup Repository Options 140
19.5 Metadata Options 144
19.6 Compression Options 144
19.7 Incremental Backup Options 147
19.8 Partial Backup and Restore Options 149
19.9 Single-File Backup Options 154
19.10 Performance / Scalability / Capacity Options 156
19.11 Message Logging Options 163
19.12 Progress Report Options 164
19.13 Encryption Options 168
19.14 Options for Working with Encrypted InnoDB Tablespaces and Encrypted Binary/Relay Logs 168
19.15 Cloud Storage Options 169
19.16 Options for Special Backup Types 174

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
--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
--cloud-access-key-id	AWS access key ID for logging onto Amazon S3.
--cloud-aws-region	Region for Amazon Web Services that mysqlbackup access for S3.
--cloud-basicauth-url	The URL for HTTP Basic Authentication for accessing Swift.
--cloud-bucket	The storage bucket for the backup image.
--cloud-buffer-size	Size of buffer for cloud operations.
--cloud-chunked-transfer	Use chunked transfer with cloud storage service.
--cloud-container	The Swift container for the backup image.
--cloud-host	The hostname for a storage service.
--cloud-identity-url	The URL of the Keystone identity service.
--cloud-object	The storage object for the backup image.
--cloud-object-key	The Amazon S3 object key for the backup image.
--cloud-password	Password for user specified by --cloud-user-id.
--cloud-proxy	Proxy address and port number for overriding the environment's default proxy settings for accessing cloud service.
--cloud-region	The Keystone region for the user specified by --cloud-user-id.
--cloud-secret-access-key	AWS secret access key.
--cloud-service	Cloud service for data backup or restoration.
--cloud-tempauth-url	The URL of the identity service for authenticating user credentials with Swift's TempAuth authentication system.
--cloud-tenant	The Keystone tenant for the user specified by --cloud-user-id.
--cloud-trace	Print trace information for cloud operations.
--cloud-user-id	User ID for accessing Swift.
--comments	Specifies comments string.
--comments-file	Specifies path to comments file.
--compress	Create backup in compressed format.
--compress-level	Specifies the level of compression.
--compress-method	Specifies the compression algorithm.
--connect_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.

Option Name	Description
--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.
--enable-cleartext-plugin	Enables the cleartext authentication plugin.
--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.
--help	Display help.
--host	Host name to connect.
--include-tables	Include in a backup or a restore tables whose names match the regular expression REGEXP.
--incremental	Specifies that the associated backup or backup-to-image operation is incremental.
--incremental-backup-dir	Specifies the location for an incremental directory backup.
--incremental-base	The specification of base backup for --incremental option.
--incremental-with-redo-log-only	Specifies the incremental backup of InnoDB tables to be based on copying redo log to the backup, without including any InnoDB data files in the backup.
--innodb_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.
--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.
--log-bin-index	Specifies the absolute path of the index file that lists all the binary log files.

Option Name	Description
--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.
--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.
--optimistic-time	Perform an optimistic backup with the value specified with the option as the optimistic time—a time after which tables that have not been modified are believed to be inactive tables.
--page-reread-count	Maximum number of page re-reads.
--page-reread-time	Wait time before a page re-read.
--password	Connection password.
--pipe	alias for –protocol=pipe.
--plugin-dir	Specifies the directory for client-side plugins.
--port	TCP portnumber to connect to.
--print-defaults	Print a list of option values supplied by defaults files and exit.
--process-threads	Specifies the number of process-threads for the backup operation.
--progress-interval	Interval between progress reports in seconds.
--protocol	Connection protocol.
--read-threads	Specifies the number of read-threads for the backup operation.
--relay-log-index	Specifies the absolute path of the index file that lists all the relay log files.
--rename	Rename a single table when it is selected by the --include-tables option to be restored

Option Name	Description
--sbt-database-name	Used as a hint to the Media Management Software (MMS) for the selection of media and policies for tape backup.
--sbt-environment	Comma separated list of environment variable assignments to be given to the SBT library.
--sbt-lib-path	Path name of the SBT library used by software that manages tape backups.
--shared-memory-base-name	It designates the shared-memory name used by a Windows server to permit clients to connect using shared memory (Windows only).
--show-progress	Instructs mysqlbackup to periodically output short progress reports known as progress indicators on its operation.
--skip-binlog	Do not include binary log files during backup, or do not restore binary log files during restore.
--skip-final-rescan	Skip the final rescan for InnoDB tables that are modified by DDL operations.
--skip-messages-logdir	Disable logging to teelog file.
--skip-relaylog	Do not include relay log files during backup, or do not restore relay log files during a restore.
--skip-unused-pages	Skip unused pages in tablespaces when backing up InnoDB tables.
--sleep	Time to sleep in milliseconds after copying each 1MB of data.
--socket	Socket file to use to connect.
--src-entry	Identifies a file or directory to extract from a single-file backup.
--ssl-ca	CA file in PEM format (implies –ssl).
--ssl-capath	CA directory (check OpenSSL docs,implies --ssl).
--ssl-cert	X509 cert in PEM format (implies --ssl).
--ssl-cipher	SSL cipher to use (implies --ssl).
--ssl-key	X509 key in PEM format (implies --ssl).
--start-lsn	Specifies the highest LSN value included in a previous backup.
--suspend-at-end	Pauses the mysqlbackup command when the backup procedure is close to ending.
--trace	Trace level of messages by mysqlbackup.
--uncompress	Uncompress a backup during an operation.
--use-tts	Enable selective backup of InnoDB tables using transportable tablespaces (TTS).
--user	Database server user name to connect.
--verbose	Print more verbose information.

Option Name	Description
--version	Display version information.
--with-timestamp	Create a subdirectory underneath the backup directory with a name formed from the timestamp of the backup operation.
--write-threads	Specifies the number of write-threads for the backup operation.

19.1 General Options

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

- The following general options also exist for the `mysql` command. Full descriptions for these options can be found in the MySQL reference manual, for example in Server Option, System Variable, and Status Variable Reference. *These options must be specified ahead of any other `mysqlbackup` options, including the rest of the general options*:
  - `--print-defaults`: Print the program argument list and exit.
  - `--no-defaults`: Don't read default options from any option file.
  - `--defaults-file=PATH`: Only read default options from the given file. It has to be the first option to be specified, if used.
  - `--defaults-extra-file=PATH`: Read this file after the global files are read.
  - `--defaults-group-suffix=str`: Also read option groups with the usual names and a suffix of `str`.
- The following options are also common between `mysqlbackup` and `mysql`, and full descriptions for them can be found in the MySQL reference manual, accessible through, e.g., Server Option, 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 general options are available for `mysqlbackup`:
  - `--verbose`: Print more verbose information.

    `--verbose` is synonymous with `--trace=1`.
  - `--debug=STRING`: Print additional debug information.

    The option accepts one or more of the following arguments (multiple values should be separated by semicolons (;)) :
    - `all`: Print additional debug information for all operations
    - `sbt`: Print additional debug information for operations using the System Backup to Tape (SBT) interface

- `page_validation`: Print debug information for validation operations.
- `hole_punch`: Print debug information for transparent page compression for backups.
- `tablespace_flags`: Print flags read from tablespaces.
- `io_file`: Print debug information for reading and writing files.
- `keyring_migration_tool`: Print debug information on keyring migration operations.

- `--force`: *(Deprecated and ignored):* 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`*

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

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

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

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

- `--error-code=`*`CODE`*

Command-Line Format	`--error-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.

- `--enable-cleartext-plugin`

  Enables the Client-Side Cleartext Pluggable Authentication. Required when using simple LDAP authentication. See Chapter 15, *Using LDAP for Server Authentication*.

- `--plugin-dir =plugin-dir`

  Specifies the directory for the client-side plugins. Required when using SASL-based LDAP authentication *AND* the client-side plugin is not in the server's plugin directory. See Chapter 15, *Using LDAP for Server Authentication* for details.

19.2 Connection Options

When `mysqlbackup` creates a backup, it sends SQL commands to a MySQL server using a database server connection. The way to create a connection is similar to what is described in Connecting to the MySQL Server Using Command Options of the MySQL 26.10 Reference Manual.

As part of the `mysqlbackup` invocation, specify the appropriate `--user`, `--password`, `--port`, and other options to connect to the MySQL server. You can specify the connection-specific MySQL client options listed below in the `[mysqlbackup]` or `[client]` sections of a MySQL configuration file, or through `mysqlbackup` command-line options (click on the option name to see its description in the mysql Client Options section of the MySQL 26.10 Reference Manual):

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 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.

- `--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`
- `--zstd-compression-level=number`

Most other connection parameters used by the `mysql` command are recognized, but silently ignored. Unknown connection parameters cause `mysqlbackup` to throw an error and quit.

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 server 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. The specified directory must be non-existent or empty, or the restore operation will fail.

  - 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 server 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. The specified directory must be non-existent or empty, or the restore operation will fail.

  - 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 server connection.

  For restores:

  - 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.*

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`*

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

Specify the path name of the file used for a single-file backup, restore, or another single-file operation.

By default, the single-file backup is streamed to standard output, so that you can pipe it directly to other commands such as a tape backup or an `ssh`-related network command. This default behavior occurs when the `--backup-image` option is omitted from the command, or when the command has `--backup-image=-`.

Except when `--backup-image=-`, if the `--backup-image` option is used but its value is not a full file path, 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 `copy-back-and-apply-log` and single-file operations other than `backup-to-image` , `mysqlbackup` takes the value of `--backup-image` as a path relative to the current working directory in which the `mysqlbackup` command is run.

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 singe file whose name is specified with the `--backup-image` option.

    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, unless it points to a location that is under *backup-dir* but is not equal to *backup-dir*/datadir, in which case mysqlbackup will throw an error and abort the operation.
- If backup_innodb_data_home_dir is a relative path, the path is taken to be relative to (that is, underneath) *backup-dir*, and it should point to *backup-dir*/datadir, or mysqlbackup will throw an error and abort the operation.
- 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.

The value for 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, unless it points to a location that is under *`backup-dir`* but is not equal to *`backup-dir/`*`datadir`, in which case `mysqlbackup` will throw an error and abort the operation.
- If `backup_innodb_log_group_home_dir` is a relative path, the path is taken to be relative to (that is, underneath) *`backup-dir`*, and it should point to *`backup-dir/`*`datadir`, or `mysqlbackup` will throw an error and abort the operation.
- An empty string (“”) for the option produces an error.

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.

- `backup_innodb_undo_directory`=*`PATH`*

  The relative or absolute directory path where separate tablespaces are created for the InnoDB undo logs during the backup. Specify the option if you want to put the undo logs at somewhere other than the default location (which is `backup_innodb_log_group_home_dir`). 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.

  The value for the parameter is derived as follows:

  - If `backup_innodb_undo_directory` is not specified, the option takes up the same value as `backup_innodb_log_group_home_dir`.
  - If `backup_innodb_undo_directory` is an absolute path, its value is used as-is, unless it points to a location that is under *`backup-dir`* but is not equal to *`backup-dir/`*`datadir`, in which case `mysqlbackup` will throw an error and abort the operation.
  - If `backup_innodb_undo_directory` is a relative path, the path is taken to be relative to (that is, underneath) *`backup-dir`* and it should point to *`backup-dir/`*`datadir`, or `mysqlbackup` will throw an error and abort the operation.
  - An empty string (“”) for the option produces an error.

  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, remove the previous backup image or backup files manually.

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 server. See Section 16.3, “Using the MySQL Enterprise Backup Logs” for details about these tables.

  Default: history logging is enabled.

- `--comments=STRING`

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

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

- `--comments-file=PATH`

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

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

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

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`

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

	`lz4`
	`lzma`
	`punch-hole`
	`none`

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 database servers, 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`: (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 suppo
```

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`*

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

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

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

- `--uncompress`

  When used with the `extract` operation, uncompresses files that are extracted from a compressed single-file backup (the option is not required *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 LSN (specified directly or indirectly by the options you use) is copied into the incremental backup.

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

Command-Line Format	`--incremental`
Type	Enumeration
Default Value	`page-track`

Valid Values	`page-track`
	`full-scan`
	`optimistic`

  Possible values for this option and their effects are explained further in the following list:

  - `page-track`: `mysqlbackup` looks for changed pages in `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.

    If page tracking cannot be used (see Incremental Backup Using Page Tracking) and `--incremental` is not set explicitly, `mysqlbackup` performs a full-scan backup instead. If this value has been set explicitly but page tracking cannot be used, the backup fails with an error.

  - `full-scan`: `mysqlbackup` scans all `InnoDB` data files in the server data directory to find pages that have been changed since the last backup, and copies them.

  - `optimistic`: `mysqlbackup` scans only 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 backups when not many tables in the database have been modified. Some restrictions apply when this option is employed; see Full-scan versus Optimistic Incremental Backup.

  `--incremental` also requires the use of either `--incremental-base` or `--start-lsn`. Only `InnoDB` tables are backed up incrementally. By default, *all* files are included in the incremental backup in their entirety. To exclude data that is not specific to *InnoDB* in an incremental backup, use `--only-innodb`.

  When `--copy-back-and-apply-log`, `--copy-back`, or `--apply-log` is used, there is no need to use `--incremental`.

- `--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 compared to backups created with the `--incremental` option; see Creating Incremental Backups Using Only the Redo Log for a discussion on their differences, and for how to perform properly a redo-log-only incremental backup.

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`

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

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

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

Type	Numeric

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

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

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

  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`*

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

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

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

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`*

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

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

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

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, besides the `ibdata*` files, have the following extensions:

- `.ARM`: ARCHIVE table metadata
- `.ARZ`: ARCHIVE table data
- `.CSM`: CSV table metadata

- `.CSV`: CSV table 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

- `--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 there is no need to copy MyISAM files.

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

  Default: backups include files from all storage engines.

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

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

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

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

  However, the option has the following limitations:

  - 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
  - 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 server 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 form of *`old_db_name`*.*`old_tb_name`* and *`new_db_name`*.*`new_tb_name`*) or not. By using fully-qualified table names, a table can be restored into a database different from its original one. If the database specified with *`new_db_name`* does not exist on the target server, it will be created during the restore process. Regular expressions are not accepted in the argument of the option.
- 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 (Deprecated).

- `--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`*

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

Identifies files or directories whose pathnames contain the STRING to be extracted from a single-file backup. This option is used with the `extract` 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`

Command-Line Format	`--dst-entry=PATH`
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`

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

For tape backups, this option can be used as a hint to the Media Management Software (MMS) for the selection of media and policies. This name has nothing to do with MySQL database names. It is a term used by the MMS. See Section 4.3.1.2, “Backing Up to Tape” for usage details.

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

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

Path name of the SBT library used by the software that manages tape backups. If this is not specified, operating system-specific search methods are used to locate `libobk.so` (UNIX) or `orasbt.dll` (Windows). See Section 4.3.1.2, “Backing Up to Tape” for usage details.

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

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

Type	String

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

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

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

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

- `--disable-manifest`

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

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`*

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

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

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

Default: currently 14.

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

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

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

If you change the number of read, write, and processing threads, you can experiment with changing this value so that it is slightly larger than the total number of threads specified by those other options. See Section 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`

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

Specifies the number of threads to use for reading data from disk. This option applies to these kinds of operations: `copy-back`, `copy-back-and-apply-log` (for directory backups only), `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 the following operations, the number of read threads is always 1 regardless of this option's setting:

- Restores of single-file backups
- `apply-log` operations and the apply-log phase of a `copy-back-and-apply-log`, `backup-and-apply-log`, or `apply-incremental-backup` operation.

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`

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

Specifies the number of threads to use for processing data, 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`

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

Specifies the number of threads to use for writing data to disk. This option applies to these kinds of operations: `copy-back`, `copy-back-and-apply-log`, `extract`, `backup`, and `backup-and-apply-log` (when applied to directory backups). For the supported operations, 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. The option is ignored for other operations like single-file backups (uses 1 write thread always), `list-image` (uses no write threads), and `validate` (uses no write threads).

If you specify a value of 0, it is silently adjusted to 1 (for operations that uses write threads). The maximum is 15. If you supply a negative value, it is silently adjusted to 15.For `apply-log` operations 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`

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

Unit	megabyte

Specify maximum memory in megabytes that can be used by `mysqlbackup`. 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, 400 for all other operations (in megabytes).

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

- `--sleep=MS`

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

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

Default: 0 (no voluntary sleeps).

- `--no-locking`

Disables 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`

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

The option is no longer supported.

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

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

Unit	millisecond

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

- `--page-reread-count=number`

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

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

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

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

Valid Values	`abort`
	`warn`
	`abort_and_remove`

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

- `abort`: Abort backup, without removing the backup directory. The disk remains full.
- `abort_and_remove`: Abort backup and remove the backup directory.
- `warn`: Write a warning message every 30 seconds and retry backup until disk space becomes available.

Default: `abort`.

- `--skip-unused-pages`

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

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

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

- `--skip-binlog`

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

  Binary log files, together with the binary log index file, are backed up and restored by default for all kinds of backups (full, incremental (see Binary Log and Relay Log Restore for exceptions), 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.:

  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.

  When an incremental backup is restored with the `--skip-binlog` option, `mysqlbackup` renames any binary log files that have already been restored onto the server by adding to them the `.old` extension.

- `--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 backed up and restored by default for all kinds of backups (full, incremental (see Binary Log and Relay Log Restore for exceptions), 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 or restoring 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`

  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`

  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 server 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.

  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*]

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

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

Accepted formats for specifying the option include:

- `now`: This includes all tables into the optimistic phase of the backup process. It is the default value for the option when no value is specified.
- *`{Number}{Unit}`*: Indicates the optimistic time as a time at a certain duration into the past. *`{Unit}`* can be any one of `years`, `months`, `hours`, and `minutes`. Some examples for option strings in this format include: `5years`, `2days`,`13months`, `23hours`, and `35minutes`.
- A date-time format in any of the following forms: `YYMMDD`, `YYYYMMDD`, `YYMMDDHHMMSS`, `YYYYMMDDHHMMSS`, `YY-MM-DD`, `YYYY-MM-DD`, `YY-MM-DD HH.MM.SS`, or `YYYYMMDDTHHMMSS` (where `T` is the character `T`).

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

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

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

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

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

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

- `--free-os-buffers`[=*NUMBER*]

Command-Line Format	`--free-os-buffers[=NUMBER]`

Type	Integer
Default Value	2
Valid Values	0 1 2 3 4 5

Default: Automatic syncing disabled.

Only on platforms supporting `posix_fadvise()`: `mysqlbackup` advises the file system with `posix_fadvise()` to free blocks from the system buffer cache, in order to minimize the impact on the system by `mysqlbackup`. The manner `posix_fadvise()` is used depends on the argument for the option:

- `0`: Do not use `posix_fadvise()` at all
- `1`, or no number given with the option: Use `posix_fadvise()` for reading and writing, and syncing files at close. This option preserves the behavior of earlier versions of `mysqlbackup`.
- `2`, or option not used : Use `posix_fadvise()` for reading or writing, depending on the operation.
- `3`: Use `posix_fadvise()` for reading and writing.
- `4`: Use `posix_fadvise()` for reading only.
- `5`: Use `posix_fadvise()` for writing only.

Default: `2`.

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`

Command-Line Format	`--messages-logdir=PATH`
Type	Directory name
Default Value	`backup_dir/meta`

Specifies the path name of an existing directory for storing the message log. If the specified directory does not exist, message logging fails and returns an error message. When this option is omitted, the default directory of *`backup_dir`*`/meta` is used, where *`backup_dir`* is the directory specified with the `--backup-dir` option.

Note

Use this option to turn on message logging for the `list-image` and `validate` operations. Message logging is turned off by default for the two operations, because they do not modify any files and a message log is usually not required for debugging them. And because the default path name of `backup_dir/meta` is not meaningful for the two operations, this option is required for both turning on message logging and for supplying the path name of a directory in which to save the log file. However, if the `--skip-messages-logdir` option is also specified, it takes precedence and message logging is skipped.

The following are some examples showing how the message logging is controlled.

This creates a log file for the `backup` operation in the directory `/home/backup_dir/meta` due to the default settings:

```
mysqlbackup -uroot --port=3306 --backup-dir=/home/backup_dir backup
```

This skips message logging for the `backup` operation:

```
mysqlbackup -uroot --port=3306 --backup-dir=/home/backup_dir \
      --skip-messages-logdir backup
```

This creates a log file for the `apply-log` operation in an existing directory named `/home/teelog_dir`, rather than the default location:

```
mysqlbackup -uroot --port=3306 --backup-dir=/home/backup_dir \
      --messages-logdir=/home/teelog_dir apply-log
```

This creates a log file for the `list-image` operation in an existing directory named `/home/teelog_dir`:

```
mysqlbackup -uroot --port=3306 --backup-image=/backup/my.mbi \
      --messages-logdir=/home/teelog_dir list-image
```

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}]`

Command-Line Format	`--show-progress[=destinations]`

Type	Enumeration

Valid Values	`stderr`
	`stdout`
	`file:FILENAME`
	`fifo:FIFONAME`
	`table`
	`variable`

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

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

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

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

- `stdout`: Progress indicators are sent to the standard output stream. A single newline character is printed after each progress indicator.
- `file:FILENAME`: Progress indicators are sent to a file. Each new progress report overwrites the file, and the file contains the most recent progress indicator followed by a single newline character.
- `fifo:FIFONAME`: Progress indicators are sent to a file system FIFO. A single newline character is printed after each progress indicator.

  Warning

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

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

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

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

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

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

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

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

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

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

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

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

  ```
  DATA of TOTAL UNIT
  ```

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

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

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

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

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

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

    For some operations such as `backup`, it is not possible to know exactly at the start of the execution how much data will be copied or processed. Therefore, the total data meter shows the estimated

amount of the total data for a backup. The estimate is updated during the execution of the command. For example:

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

is followed by:

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

when 100MB of data is added on the server.

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

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

  ```
  compression: 40%
  ```

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

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

  ```
  compression: -
  ```

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

  ```
  state: Copying InnoDB data
  ```

  ```
  state: Waiting for locks
  ```

  ```
  state: Copying system tablespace
  ```

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

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

  ```
  state: Completed
  ```

Here are some examples of progress indicators with different meters:

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

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

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

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

Command-Line Format	`--progress-interval=SECONDS`
Type	Numeric
Default Value	`2`
Minimum Value	`1`

Maximum Value	`100000`
Unit	second

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

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`*

Command-Line Format	`--key=KEY`
Type	String

The symmetric key for encryption and decryption of a backup image. It should be a 256-bit key, encoded as a string of 64 hexadecimal digits. See Chapter 10, *Encryption for Backups* on how to create a key. The option is incompatible with the `--key-file` option.

- `--key-file=`*`PATH`*

Command-Line Format	`--key-file=FILE`
Type	File name

The pathname to file that contains a 256-bit key, encoded as a string of 64 hexadecimal digits, for encryption and decryption of a backup image. The option is incompatible with the `--key` option.

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

MySQL Enterprise Backup supports encrypted InnoDB tablespaces and 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`*]

Command-Line Format	`--encrypt-password=STRING`
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 or component 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 `component_keyring_encrypted_file` keyring component, the password supplied with the option must match the keyring file encryption password that has been set on the server with the `component_keyring_encrypted_file.cnf` file. If the server uses the `keyring_hashicorp` plugin, use the option to supply the HashiCorp Vault AppRole authentication secret ID, which was the value of `keyring_hashicorp_secret_id` on the server to be backed up.

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 5.2, "Restoring a Backup from Cloud Storage to a MySQL Server", for more information and instructions on using cloud storage with MySQL Enterprise Backup.

- *Options used for all cloud services:*
  - `--cloud-service=`*`SERVICE`*

    Cloud service for data backup or restoration. Various types of cloud storage services are supported by `mysqlbackup`, represented by the following values for the option:

    - `OCI`: Oracle Cloud Infrastructure Object Storage
    - `openstack`: OpenStack Swift or compatible object storage services
    - `s3`: Amazon Simple Storage Service (S3) or compatible storage service
    - `GCP`: GCP object storage
  - `--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 for Oracle Cloud Infrastructure (OCI) Object Storage:*

  - `--cloud-object=OCI_OBJECT`

    The storage object for the backup image. Note that names of objects within the same bucket have to be unique.

  - `--cloud-par-url=OCI_PAR-URL`

    The Pre-Authenticated Request (PAR) URL for OCI Object Storage. For a backup to OCI Object Storage, it is the PAR URL for the storage bucket; for restore and other operations on an object stored on OCI, it is the PAR URL for the object.

- *Options used for OpenStack Swift Object Storage:*
  - `--cloud-object=SWIFT_OBJECT`

    The storage object for the backup image. Note that names of objects within the same container have to be unique.
  - `--cloud-container=SWIFT_CONTAINER`

    The Swift container for the backup image.
  - `--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-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={true|false}`

    Use chunked transfer. When `--cloud-service=openstack`, backups are always transferred and stored as Dynamic Large Objects (DLOs), for which multiple file segments are considered as a

single file. The maximum number of segments a backup can have is determined by the object storage service, and the maximum size of the segments is controlled by this option.

If the option is set to true, `mysqlbackup` uses chunked transfer encoding to transfer the data. A backup larger than the value of `--cloud-chunk-size` is split into multiple segments.

If the option is set to false, `mysqlbackup` uploads the backup in segments in the size of the buffer.

Default: false, when `--cloud-service=openstack`

> **Warning**
>
> Set the option to true only if chunked transfer is supported by your cloud storage; otherwise, the `mysqlbackup` operation may fail.

- `--cloud-chunk-size=SWIFT_CHUNK-SIZE`

  The size of a chunk in megabytes if chunked transfer is enabled. This option is ignored if chunked transfer is disabled.

  Minimum value: 64

  Maximum value: 3072 on 32-bit machines, 5120 on 64-bit machines

  Default value: 2048

> **Note**
>
> 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 for Amazon S3 and S3-compatible services:*
  - `--cloud-host=S3_HOSTNAME`

    The hostname for an S3-compatible storage service.

    Default: `s3.amazonaws.com`
  - `--cloud-bucket=S3_BUCKET`

    The storage bucket on an S3-compatible storage service 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 S3 object key for the backup image.
  - `--cloud-access-key-id=S3_KEY-ID`

    Access key ID for logging onto an S3-compatible storage service.
  - `--cloud-secret-access-key=S3_ACCESS-KEY`

    Secret access key for the access key id specified with `--cloud-access-key-id`.
  - `--cloud-aws-region=S3_REGION`

    Region for the web service that `mysqlbackup` accesses.

- *Options used for GCP object storage :*
  - `--cloud-host=HOSTNAME`

    The hostname for the storage service.

    Default: `storage.googleapis.com`, when `--cloud-service=GCP`
  - `--cloud-bucket=BUCKET`

    The storage bucket for the backup image.
  - `--cloud-object=OBJECT`

    The storage object for the backup image. Note that names of objects within the same bucket have to be unique.
  - `--cloud-access-key=ACCESS-KEY`

    Access key for the cloud account. The option is mandatory for a backup operation. If it is not provided for a restore operation, `mysqlbackup` assumes the backup image is a public object and can be accessed or downloaded without a key; if the object is not public, the restore operation fails.
  - `--cloud-secret-key=SECRET-KEY`

    Secret key for the cloud account specified with `--cloud-access-key`. The option is mandatory for a backup operation.
  - `--cloud-chunked-transfer={true|false}`

    Use chunked transfer.

    If the option is set to true, `mysqlbackup` uses chunked transfer encoding to transfer the data. A backup larger than the value of `--cloud-chunk-size` is split into multiple segments.

    If the option is set to false, `mysqlbackup` uploads the backup in segments in the size of the buffer.

    Default: true, when `--cloud-service=GCP`
  - `--cloud-chunk-size=CHUNK-SIZE`

    The size of a chunk in megabytes if chunked transfer is enabled. This option is ignored if chunked transfer is disabled.

    Minimum value: 64

    Maximum value: 3072 on 32-bit machines, 5120 on 64-bit machines

    Default value: 2048

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.

- `--replica-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_replica_info` inside the backup directory, containing a `CHANGE REPLICATION SOURCE TO` 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 REPLICATION SOURCE TO` command with the binary log position saved in the `ibbackup_replica_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 `--replica-info` option to have the file `backup_gtid_executed.sql` generated and included in the backups.

The option is an alias of `--slave-info`, which has been decprecated .

- `--slave-info`

(Deprecated; use `--replica-info` instead) 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-replica-backup-timeout=SECONDS`

For a replication setup, the option specifies the time (in seconds) `mysqlbackup` will wait for `Replica_open_temp_tables` to become “0” (which happens 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 a 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 `Replica_open_temp_tables=0` becomes true within the duration set by `--safe-replica-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-replica-backup-timeout`, or retry at a time when fewer temporary tables are being used.

Default: 300

Warning

Proper setting of this value depends on the use case, and it can vary a lot according to the situation. Setting the value for this option either too high or too low will affect adversely the performance of the backup operation:

- *Too high:* If you need to wait for a long time for there to be no more temporary tables, the chance is that the change rate for your database server is quite high, which means the amount of redo log data to be included in the backup will be large and the restore time for the backup will be long. In such a case, it would have been better to have let `mysqlbackup` timeout and then restart the backup operation, so the tables are copied in their final states. It is therefore not helpful to set a high timeout value for the option. As a very general rule of thumb: even for busy databases that use many temporary tables, do not set the value to more than a few hours.
- *Too low:* Setting the wait time value too low would make the backup process time out very easily and when that happens, the process has to be restarted. With a repeating cycle of restarts, the backup might then take a long time to complete, and resources used on the failed backups will be wasted. As a very general rule of thumb, do not set the timeout to below the default value of 300s.

Even though the issue described above with temporary tables does not exist for a row-based replication (RBR) setup (for which temporary tables are not replicated onto the replica), any replica backups still wait for temporary tables to be all closed as described above, because the mode of replication could potentially be switched from RBR to statement-based replication (SBR) or mixed-mode replication at any time. Users who are certain that SBR is not occurring for the replica can set `--safe-replica-backup-timeout=0`, with which `mysqlbackup` will not check for any open temporary tables before finishing the backup.

The option is an alias of `--safe-slave-backup-timeout`, which has been decprecated .

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

  (Deprecated ; use `--safe-replica-backup-timeout` instead) For a 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 a replica for details on how `mysqlbackup` deals with temporary tables on a replica server.

In addition, `mysqlbackup` also runs an initial check at the beginning of a replica backup to see if `Slave_open_temp_tables=0` becomes true within the duration set by `--safe-slave-backup-timeout`. If it does not, `mysqlbackup` takes it as an early sign that before the backup is completed, some temporary tables are likely to remain open after the timeout limit is exceeded; `mysqlbackup` then throws an error, instead of continuing with the backup. When that happens, you can either restart the backup with a higher value for `--safe-slave-backup-timeout`, or retry at a time when fewer temporary tables are being used.

Default: 300

Warning

Proper setting of this value depends on the use case, and it can vary a lot according to the situation. Setting the value for this option either too high or too low will affect adversely the performance of the backup operation:

- *Too high:* If you need to wait for a long time for there to be no more temporary tables, the chance is that the change rate for your database server is quite high, which means the amount of redo log data to be included in the backup will be large and the restore time for the backup will be long. In such a case, it would have been better to have let `mysqlbackup` timeout and then restart the backup operation, so the tables are copied in their final states. It is therefore not helpful to set a high timeout value for the option. As a very general rule of thumb: even for busy databases that use many temporary tables, do not set the value to more than a few hours.
- *Too low:* Setting the wait time value too low would make the backup process time out very easily and when that happens, the process has to be restarted. With a repeating cycle of restarts, the backup might then take a long time to complete, and resources used on the failed backups will be wasted. As a very general rule of thumb, do not set the timeout to below the default value of 300s.

Even though the issue described above with temporary tables does not exist for a row-based replication (RBR) setup (for which temporary tables are not replicated onto the replica), any replica backups still wait for temporary tables to be all closed as described above, because the mode of replication could potentially be switched from RBR to statement-based replication (SBR) or mixed-mode replication at any time. Users who are certain that SBR is not occurring for the replica can set `--safe-slave-backup-timeout=0`, with which `mysqlbackup` will not check for any open temporary tables before finishing the backup.

- `--suspend-at-end`

  This option pauses the `mysqlbackup` command when the backup procedure is close to ending. It creates a file called `ibbackup_suspended` in the backup log group home directory and waits until you delete that file before proceeding. This option is useful to customize locking behavior and backup of non-InnoDB files through custom scripting.

  All 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.

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

Command-Line Format	`--exec-when-locked="utility arg1 arg2 ..."`
Type	String

The specified *`utility`* is executed when all non-InnoDB 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_command
```

Or on Windows systems:

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

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

Chapter 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 the following options in the `[mysqld]` group in the configuration file to detect parameters related to the server when it is not connected to it (for example, when restoring a non-TTS backup):

- `datadir`
- `innodb_data_home_dir`
- `innodb_log_group_home_dir`
- `innodb_undo_directory`
- `log-bin`
- `relay_log`

If any of these options is also specified in the `[mysqlbackup]` section, the value there takes precedence over the value in the `[mysqld]` section. Options other than the ones listed above should be specified under the `[mysqlbackup]` or `[client]` section.

Within a `mysqlbackup` option name, dashes (`-`) and underscores (_) may usually be used interchangeably, similar to `mysqld` parameters that use this same convention (notice that `log-bin` is one exception; 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_page_size=16384
innodb_checksum_algorithm=crc32
innodb_buffer_pool_filename=ib_buffer_pool
innodb_undo_tablespaces=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 185
B Limitations of MySQL Enterprise Backup 187
C Compatibility Information for MySQL Enterprise Backup 191
    C.1 Supported Platforms 191
    C.2 Cross-Platform Compatibility 191
    C.3 Compatibility with MySQL Versions 191
    C.4 Compatibility with Older MySQL Enterprise Backup 192
D Backup History Table Update 193
E SBT Backup History Table Update 195
F Backup Progress Table Update 197
MySQL Enterprise Backup Glossary 199

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 26.10.0 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 server 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 26.10.0 support?

See Section C.3, “Compatibility with MySQL Versions” for details of compatibility between different releases of MySQL Enterprise Backup and MySQL Server.

A.2: What is the big `ibdata` file that is in all the backups?

You might find your backup data taking more space than expected because of a large file with a name such as `ibdata1`. This file represents the InnoDB system tablespace, which grows but never shrinks as a database server 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 databases, 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.

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 server on one OS platform and restore it on another one using MySQL Enterprise Backup?

See Section C.2, “Cross-Platform Compatibility” for details.

A.7: What if I have included the binary log or relay log in my backup but do not want to restore it?

If you want to skip the restore of the binary log, relay log, or both during a restore, use the `--skip-binlog` option, the `--skip-relaylog` option, or both with your `copy-back` or `copy-back-and-apply-log` command.

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 server during the backup process can be applied.

Appendix B Limitations of MySQL Enterprise Backup

Please refer to the MySQL Enterprise Backup 26.10 Release Notes for a list of fixed bugs for `mysqlbackup`. Here is a list of limitations of MySQL Enterprise Backup:

- 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) 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 or 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 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.
- 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
- A compressed directory backup fails when a general tablespace bears the same basename as the database server's system tablespace (usually `ibdata1`) and exists in the same directory with it (usually the server's data directory). A compressed single-file backup created under the same situation will be corrupted, and cannot be restored. To avoid the problem, the server administrator should not put into the same directory the system tablespace and a general tablespace of the same basename; if that is unavoidable, do not perform a compressed backup for the database server.
- When working with a replication set up whose source server also belongs to a separate Group Replication setup, over time, create backups consistently either from the source or the replica, but not from both. Otherwise, there will be conflicts between the `id` values generated by the source and the replica, causing backups to fail.
- A backup fails if the name of any database is the same as the name of any undo tablespace. For backups to be successful, the database administrator should avoid giving any database and undo

tablespace the same name (for example, using the default undo tablespace name `undo_001` to name a database), or the database should be renamed before a backup.

Appendix C Compatibility Information for MySQL Enterprise Backup

Table of Contents

C.1 Supported Platforms 191
C.2 Cross-Platform Compatibility 191
C.3 Compatibility with MySQL Versions 191
C.4 Compatibility with Older MySQL Enterprise Backup 192

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

C.1 Supported Platforms

See Supported Platforms: MySQL Database (MySQL platform support evolves over time; please refer to the page for the latest updates).

C.2 Cross-Platform Compatibility

MySQL Enterprise Backup is by nature cross-platform compatible when running on the Linux and Windows operating systems: backups on a Linux machine can, in principle, be restored on a Windows machine, and vice versa. However, because Identifier Case Sensitivity is dependent on the case sensitivity of the underlying operating system (which varies across platforms), data transfer problems might occur due to letter case issues with the database or table names. Therefore, to ensure cross-platform compatibility, the following two conditions must be fulfilled:

- Both the backed-up and the restored servers have `lower_case_table_names`=1 (which means both servers were initialized with that setting, as `lower_case_table_names` cannot be changed after server initialization), so that all table and database names are stored in lowercase on disk and name comparisons are not case-sensitive.
- All applications that connect to the backed-up and the restored servers are always consistent on their use of letter cases when referring to the database and table names.

Warning

Meeting these conditions does not guarantee cross-platform compatibility. Users should perform tests on their systems to make sure cross-platform restores work as desired.

C.3 Compatibility with MySQL Versions

MySQL Enterprise Backup 26.10.0 supports the MySQL Server 26.10.x series for back up and restore. It is incompatible with other versions of MySQL Server, for which you should use the MySQL Enterprise Backup of the same release number, or consult the MySQL Enterprise Backup manual of the series for more information.

Also see important information in Section 5.4, “Restoring a Backup with a Database Server Upgrade or Downgrade”.

C.4 Compatibility with Older MySQL Enterprise Backup

MySQL Enterprise Backup 26.10.0 works with backups created by the MySQL Enterprise Backup 26.10.x series. It does not work with backups created by any other MySQL Enterprise Backup series.

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 26.10 tries to perform its first full backup on a database server, 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. 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 26.10 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 comman
mysqlbackup failed with errors!
```

These privileges are no longer needed after the first full backup has been performed by MySQL Enterprise Backup 26.10, 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 26.10 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 26.10 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 co
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 26.10, 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 26.10 tries to perform its first full backup on a database server, 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. 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 26.10 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_old,
mysql.backup_progress_new TO mysql.backup_progress': 1142, ALTER command denied to user 'mysqlbackup'@'loca
for table 'backup_progress'
```

These privileges are no longer needed after the first full backup has been performed by MySQL Enterprise Backup 26.10, 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.

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.

MySQL Enterprise Backup
: A licensed products that performs **hot backups** of MySQL database servers. 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 InnoDB.

B

backup
: The process of copying some or all table data and metadata from a MySQL instance, for safekeeping. Can also refer to the set of copied files. This is a crucial task for DBAs. The reverse of this process is the **restore** operation.

With MySQL, **physical backups** are performed by the **MySQL Enterprise Backup** product, and **logical backups** are performed by the `mysqldump` command. These techniques have different characteristics in terms of size and representation of the backup data, and speed (especially speed of the restore operation).

Backups are further classified as **hot**, **warm**, or **cold** depending on how much they interfere with normal database server operation. (Hot backups have the least interference, cold backups the most.)
See Also 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.

binlog
An informal name for the **binary log** file. For example, you might see this abbreviation used in e-mail messages or forum discussions.

C

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, InnoDB, restore.

compression level
A setting that determines how much **compression** to apply to a compressed backup. This setting ranges from 0 (none), 1 (default level when compression is enabled) to 9 (maximum). The amount of compression for a given compression level depends on the nature of your data values. Higher compression levels do impose additional CPU overhead, so ideally you use the lowest value that produces a good balance of compression with low CPU overhead.
See Also compression.

configuration file
The file that holds the startup options of the MySQL server and related products and components. Often referred to by its default file name, **my.cnf** on Linux, Unix, and macOS systems, and **my.ini** on Windows systems. The **MySQL Enterprise Backup** stores its default configuration settings in this file, under a `[mysqlbackup]` section. For convenience, MySQL Enterprise Backup can also read settings from the `[client]` section, for configuration options that are common between MySQL Enterprise Backup and other programs that connect to the MySQL server.
See Also my.cnf, my.ini.

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 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.

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**.

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

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 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, 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.

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 system tablespace, 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 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**.

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 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.

L

locking

See Also 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, incremental backup, redo log.

M

.MRG file

A file containing references to other tables, used by the `MERGE` storage engine. Files with this extension are always included in backups produced by the `mysqlbackup` command of the **MySQL Enterprise Backup** product.

.MYD file

A file that MySQL uses to store data for a MyISAM table.
See Also .MYI file.

.MYI file

A file that MySQL uses to store indexes for a MyISAM table.
See Also .MYD file.

manifest

The record of the environment (for example, command-line arguments) and data files involved in a backup, stored in the files `meta/backup_create.xml` and `meta/backup_content.xml`, respectively. This data can be used by management tools during diagnosis and troubleshooting procedures.

master

See source.

media management software

A class of software programs for managing backup media, such as libraries of tape backups. One example is **Oracle Secure Backup**. Abbreviated **MMS**.
See Also Oracle Secure Backup.

my.cnf

The typical name for the MySQL **configuration file** on Linux, Unix, and macOS systems.
See Also configuration file, my.ini.

my.ini

The typical name for the MySQL **configuration file** on Windows systems.
See Also configuration file, my.cnf.

MyISAM

A MySQL storage engine, formerly the default for new tables. In MySQL 5.5 and higher, **InnoDB** becomes the default storage engine. MySQL Enterprise Backup can back up both types of tables, and tables from other storage engines also. The backup process for InnoDB tables (**hot backup**) is less disruptive to database operations than for MyISAM tables (**warm backup**).

See Also InnoDB, warm backup.

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 server configuration information. Files with this extension are always included in backups produced by the backup operations of the **MySQL Enterprise Backup** product.

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.

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.

replica

In a **replication** configuration, a database server that receives updates from a **source** server. Typically used to service user queries, to minimize the query load on the source server. With **MySQL Enterprise Backup**, you might take a backup on one server, and restore on a different system to create a new replica server with the data already in place. You might also back up data from a replica server rather than the source, to minimize any slowdown of the overall system.
See Also replication, source.

replication

A common configuration for MySQL deployments, with data and DML operations from a **source** server synchronized with a set of **replica** servers. With MySQL **Enterprise Backup,** you might take a backup on one server, and restore on a different system to create a new replica server with the data already in place. You might also back up data from a replica server rather than the source, to minimize any slowdown of the overall system.
See Also replica, source.

repository
We distinguish between the **server repository** and the **backup repository**.
See Also backup repository, server repository.

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

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

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

S

SBT
Acronym for **system backup to tape**.
See Also system backup to tape.

selective backup
Another name for **partial backup**
See Also selective restore.

selective restore
Another name for **partial restore**
See Also 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 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.

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 server, 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 data dictionary, file format, ibdata file, tablespace.

T

.TRG file
: A file containing **trigger** parameters. Files with this extension are always included in backups produced by the `mysqlbackup` command of the **MySQL Enterprise Backup** product.

table
: Although a table is a distinct, addressable object in the context of SQL, for **backup** purposes we are often concerned with whether the table is part of the **system tablespace**, or was created under the **file-per-table** setting and so resides in its own **tablespace**.
See Also backup, system tablespace, tablespace.

tablespace
: For **InnoDB** tables, the file that holds the data and indexes for a table. Can be either the **system tablespace** containing multiple tables, or a table created with the **file-per-table** setting that resides in its own tablespace file.
See Also InnoDB, system tablespace.

transportable tablespace
: A feature that allows a **tablespace** to be moved from one instance to another. Traditionally, this has not been possible for InnoDB tablespaces because all table data was part of the **system tablespace**. In MySQL 5.6 and higher, the `FLUSH TABLES ... FOR EXPORT` syntax prepares an InnoDB table for copying to another server; running `ALTER TABLE ... DISCARD TABLESPACE` and `ALTER TABLE ... IMPORT TABLESPACE` on the other server brings the copied data file into the other instance. A separate `.cfg` file, copied along with the **.ibd file**, is used to update the table metadata (for example the **space ID**) as the tablespace is imported. See Importing InnoDB Tables for usage information.

Use the `--use-tts` option to create a backup with transportable tablespace. See also Section 5.1.5, “Restoring Backups Created with the `--use-tts` Option”.

TTS
: Short form for **transportable tablespace**.
See Also 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, transportable tablespace.

W

warm backup

A **backup** taken while the database server 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.

Index

Symbols

--log-bin option, 137
--relay-log option, 138

A

MySQL Enterprise Backup, 199
Antelope, 117
apply, 199
apply-incremental-backup option, 121
--apply-log option, 121
.ARM file, 199
.ARZ file, 199
authentication
  LDAP, 105

B

backup, 199
backup directory, 141, 199
backup option, 120
backup repository, 199
backup-and-apply-log option, 120
--backup-dir option, 141
backup-dir-to-image option, 125
backup-image option, 140
backup-my.cnf, 199
backup-my.cnf file, 7
backup-to-image option, 120, 154
backups
  cold, 5
  compressed, 5, 56, 60, 144
  containing encrypted InnoDB tablespaces, 75
  controlling overhead, performance, and scalability, 156
  differential, 50
  encrypted, 89
  full, 49
  hot, 5
  incremental, 5, 50, 147
  InnoDB tables only, 117
  logical, 5
  message logging, 163
  optimistic, 60
  parallel, 5
  partial, 57, 149
  physical, 5
  prepared, 7, 70
  preparing to restore, 70
  progress report, 164
  raw, 7, 70
  scheduled, 63
  single-file, 5, 44
  streaming, 5, 46
  to cloud, 47
  to tape, 47
    with Oracle Secure Backup, 91
  troubleshooting, 107
  uncompressed, 5, 59
  verifying, 40
  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, 193
backup_progress table, 109
  update, 197
backup_sbt_history table
  update, 195
backup_variables.txt file, 7
Barracuda, 117
benchmarking, 97
binary log, 71
binlog, 200

C

cloud backups, 47
--cloud-access-key-id option, 173
--cloud-access-key option, 174
--cloud-aws-region option, 173
--cloud-basicauth-url, 171
--cloud-bucket option, 173
--cloud-buffer-size option, 170
--cloud-ca-info option, 170
--cloud-ca-path option, 170
--cloud-chunk-size, 172, 174
--cloud-host option, 173
--cloud-chunked-transfer, 171
--cloud-identity-url, 171
--cloud-object, 170, 171
--cloud-par-url, 170
--cloud-object-key option, 173
--cloud-password, 171
--cloud-proxy option, 170
--cloud-region, 171
--cloud-secret-access-key option, 173
--cloud-secret-key option, 174
--cloud-service option, 169
--cloud-tempauth-url, 171
--cloud-tenant, 171
--cloud-trace option, 170
--cloud-user-id, 171
cold backup, 5
command-line tools, 5

commands, mysqlbackup, 119
--comments option, 144
--comments-file option, 144
comments.txt file, 7, 144
--compress option, 56, 144
--compress-level option, 56, 146
--compress-method option, 145
compressed backups, 5, 56, 60, 144
compression, 200
compression level, 200
configuration file, 200
configuration options, 179
connection options, 136
copy-back option, 65, 123
copy-back-and-apply-log option, 41, 122
corruption problems, 108
--counter-container, 171
crash recovery, 70, 200
cron jobs, 63
.CSM file, 7
.CSV file, 7

D

data dictionary, 200
database, 201
datadir directory, 7
--datadir option, 137
--data_home_dir option, 139
--decrypt option, 168
decryption, 89
--disable-manifest option, 156
disk storage for backup data, 5, 46
distributed file system, 64
--dst-entry option, 155

E

--enable-cleartext-plugin option, 136
--encrypt option, 168
--encrypt-password option, 169
encrypted backups, 89
encrypted InnoDB tables, 75
encrypted InnoDB tablespaces, 75
encrypted undo tablespaces, 75
encryption, 89
--error-code option, 107, 136
exclude, 201
--exclude-tables option, 57, 150
--exec-when-locked option, 178
exit codes, 107, 136
extract, 201
extract option, 125, 154

F

FAQ, 185
file format, 201
files backed up, 7
frequently asked questions, 185
full backup, 49, 201
full-scan incremental backup, 54

G

GRANT statement, 32
group replication, 87

H

history table, 201
hot backup, 5

I

ibbackup_logfile file, 7
.ibd file, 7, 201
ibdata file, 7, 201
ibreset command, 108
.bz file, 7
.ibz file, 7
ib_logfile file, 7
image, 202
image-to-backup-dir option, 125, 154, 154
image_files.xml file, 7, 111
include, 202
--include option, 57, 153
--include-tables option, 57, 149
incremental backup, 5, 147, 202
    full scan versus optimistic, 54
    using page tracking, 52
--incremental option, 147
--incremental-backup-dir option, 149
--incremental-base option, 148
--incremental-with-redo-log-only option, 148
InnoDB, 202
InnoDB tables, 5, 7, 117, 117
    compressed backup feature, 56
    incremental backup feature, 50
installing MySQL Enterprise Backup, 23
instance, 202

K

--key option, 168
--key-file option, 168
keyring components, 75

L

LDAP authentication
    --enable-cleartext-plugin option, 136
    --plugin-dir option, 136

--limit-memory option, 159
list-image option, 125, 154
--lock-wait-timeout option, 159
locking, 202
log, 7, 120, 202
--log-bin-index, 138
logical backup, 5, 202
logs
  of backup operations, 109, 193, 195, 197
LSN, 50, 147, 203

M

manifest, 7, 111, 156, 203
media management software, 203
media management software (MMS) products, 91
MEMORY tables, 63
message logging, 163
meta directory, 7
MMS products, 91
.MRG file, 203
my.cnf, 203
my.ini, 203
.MYD file, 7
.MYD file, 203
.MYI file, 7
.MYI file, 203
.sdi file, 7
MyISAM, 203
MyISAM tables, 117
mysqlbackup, 117, 204
  and media management software (MMS) products, 91
  commands, 119
  configuration options, 179
  examples, 44
  files produced, 7
  modes of operation, 119
  options, 129
  overview, 5
  required privileges, 32
  using, 31
mysqlbinlog command, 71
mysqldump, 63, 204

N

--no-history-logging option, 144
--no-locking option, 159
non-TTS backup, 204
--number-of-buffers option, 156

O

--on-disk-full option, 160
--only-innodb option, 151
--rename option, 152
--only-known-file-types option, 150
.opt file, 204
optimistic backup, 60, 162, 162
optimistic incremental backup, 54, 147
--optimistic-busy-tables, 162
--optimistic-time, 162
options, mysqlbackup, 129
  connection, 136
  for cloud storage, 169
  for compression, 144
  for controlling backup overhead, performance, and scalability, 156
  for controlling message logging, 163
  for controlling progress reporting, 164
  for encrypted InnoDB tablespaces, 168
  for encryption, 168
  for generating metadata, 144
  for incremental backups, 147
  for partial backups, 149
  for single-file backups, 154
  for special types of backups, 174
  general options, 134
  in configuration files, 179
  layout of backup files, 140
  layout of database files, 137
  options in common with mysql, 134
Oracle Secure Backup, 204
OS user, 36
OSB, 204
other operations, 126

P

--page-reread-count option, 160
--page-reread-time option, 160
.par file, 204
parallel backup, 97, 100, 204
parallel backups, 5
partial backup, 57, 149
partial restore, 68
performance
  of backups, 97
  of restores, 100
performance of backup operations, 5
permissions, 36
physical backup, 5, 204
--plugin-dir option, 136
point in time, 205
point-in-time recovery, 71
posix_fadvise() system call, 5
prepared backup, 7, 70, 205
print-message option, 126
privileges, 32
--process-threads option, 157

progress indicator, 164
progress table, 205
--progress-interval, 168

R

RAID, 97, 100
raw backup, 7, 70, 205
--read-threads option, 157
redo log, 205
redo log archiving, 79
regular expression, 205
--relay-log-index, 138
--free-os-buffers, 163
replica, 81, 83, 205
replication, 81, 83, 84, 205
repository, 206
restore, 206
restoring a backup, 65
  at original location, 41
  backup created with the --use-tts option, 69
  examples, 65
  mysqlbackup options, 121
  partial restore, 68
  point-in-time recovery, 71
  preparation, 70
  restore external tablespaces at different locations, 70
row format, 206

S

--safe-replica-backup-timeout, 175
--safe-slave-backup-timeout, 176
SBT, 206
  backup history table update, 195
--sbt-database-name option, 155
--sbt-environment option, 156
--sbt-lib-path option, 155
selective backup, 206
selective restore, 206
server, 206
server repository, 206
--show-progress, 165
single-file backup, 5, 44, 124, 154, 206
--skip-binlog, 161
--skip-final-rescan, 161
--no-redo-log-archive, 161
--skip-relaylog, 161
--skip-unused-pages, 160
--replica-info option, 175
--slave-info option, 175
--sleep option, 159
source, 84, 206
space for backup data, 5
--src-entry option, 154
--start-lsn option, 149
storage access network, 64
streaming, 46, 207
streaming backups, 5
--suspend-at-end option, 177
system backup to tape, 207
system tablespace, 7, 207

T

table, 207
Table-Level Recovery (TLR), 68
tablespace, 207
tablespace tracker file, 18
tape backups, 47, 91
transportable tablespace, 207
.TRG file, 207
troubleshooting for backups, 107
TTS, 207
TTS backup, 208

U

--uncompress option, 147
uncompressed backups, 5, 59
--use-tts option, 151
using with MySQL Enterprise Firewall, 103

V

validate option, 124
validating a backup, 123
verifying a backup, 40

W

warm backup, 5, 208
what is new, 25
--with-timestamp option, 143
--write-threads option, 158