Percona XtraBackup 2.4 Documentation

Release 2.4.20

Percona LLC and/or its affiliates

Apr 14, 2020

CONTENTS

Percona XtraBackup is an open-source hot backup utility for *MySQL* - based servers that doesn't lock your database during the backup.

It can back up data from *InnoDB*, *XtraDB*, and *MyISAM* tables on *MySQL* 5.1[1], 5.5, 5.6 and 5.7 servers, as well as *Percona Server for MySQL* with *XtraDB*.

For a high-level overview of many of its advanced features, including a feature comparison, please see *About Percona XtraBackup*.

Whether it is a 24x7 highly loaded server or a low-transaction-volume environment, *Percona XtraBackup* is designed to make backups a seamless procedure without disrupting the performance of the server in a production environment. Commercial support contracts are available.

Important: *Percona XtraBackup* 2.4 does not support making backups of databases created in *MySQL* 8.0, *Percona Server for MySQL* 8.0 or *Percona XtraDB Cluster* 8.0.

---

[1] Support for InnoDB 5.1 builtin has been removed in *Percona XtraBackup* 2.1

Part I

Introduction

CHAPTER

ONE

ABOUT PERCONA XTRABACKUP

Percona XtraBackup is the world's only open-source, free *MySQL* hot backup software that performs non-blocking backups for *InnoDB* and *XtraDB* databases. With *Percona XtraBackup*, you can achieve the following benefits:

- Backups that complete quickly and reliably
- Uninterrupted transaction processing during backups
- Savings on disk space and network bandwidth
- Automatic backup verification
- Higher uptime due to faster restore time

Percona XtraBackup makes *MySQL* hot backups for all versions of *Percona Server for MySQL*, *MySQL*, and *MariaDB*. It performs streaming, compressed, and incremental *MySQL* backups.

Important: *Percona XtraBackup* is not compatible with *MariaDB* 10.3 and later.

Percona XtraBackup works with *MySQL*, *MariaDB*, and *Percona Server for MySQL*. It supports completely non-blocking backups of *InnoDB*, *XtraDB*, and *HailDB* storage engines. In addition, it can back up the following storage engines by briefly pausing writes at the end of the backup: *MyISAM*, *Merge*, and *Archive*, including partitioned tables, triggers, and database options.

Important: *Percona XtraBackup* 2.4 does not support the MyRocks or TokuDB storage engines.

Percona's enterprise-grade commercial MySQL Support contracts include support for *Percona XtraBackup*. We recommend support for critical production deployments.

MySQL Backup Tool Feature Comparison

Features	Percona XtraBackup	MySQL Enterprise backup
License	GPL	Proprietary
Price	Free	Included in subscription at $5000 per Server
Streaming and encryption formats	Open source	Proprietary
Supported *MySQL* flavors	*MySQL*, *Percona Server for MySQL*, *MariaDB*, *Percona XtraDB Cluster*[9], *MariaDB Galera Cluster*	*MySQL*

Continued on next page

Table 1.1 – continued from previous page

Features	Percona XtraBackup	MySQL Enterprise backup
Supported operating systems	Linux	Linux, Solaris, Windows, OSX, FreeBSD.
Non-blocking InnoDB backups[1]	Yes	Yes
Blocking MyISAM backups	Yes	Yes
Incremental backups	Yes	Yes
Full compressed backups	Yes	Yes
Incremental compressed backups	Yes	
Fast incremental backups[2]	Yes	
Incremental backups with archived logs feature in Percona Server	Yes	
Incremental backups with REDO log only		Yes
Backup locks[8]	Yes	
Encrypted backups	Yes	Yes[3]
Streaming backups	Yes	Yes
Parallel local backups	Yes	Yes
Parallel compression	Yes	Yes
Parallel encryption	Yes	Yes
Parallel apply-log	Yes	
Parallel copy-back		Yes
Partial backups	Yes	Yes
Partial backups of individual partitions	Yes	
Throttling[4]	Yes	Yes
Backup image validation		Yes
Point-in-time recovery support	Yes	Yes
Safe slave backups	Yes	
Compact backups[5]	Yes	
Buffer pool state backups	Yes	
Individual tables export	Yes	Yes[6]
Individual partitions export	Yes	
Restoring tables to a different server[7]	Yes	Yes
Data & index file statistics	Yes	
InnoDB secondary indexes defragmentation	Yes	
`rsync` support to minimize lock time	Yes	
Improved `FTWRL` handling	Yes	
Backup history table	Yes	Yes
Backup progress table		Yes
Offline backups		Yes
Backup to tape media managers		Yes
Cloud backups support		Amazon S3
External graphical user interfaces to backup/recovery	Zmanda Recovery Manager for MySQL	MySQL Workbench, MySQL Enterprise Monitor

What are the features of Percona XtraBackup?

Here is a short list of *Percona XtraBackup* features. See the documentation for more.

- Create hot *InnoDB* backups without pausing your database
- Make incremental backups of *MySQL*
- Stream compressed *MySQL* backups to another server
- Move tables between *MySQL* servers on-line
- Create new *MySQL* replication slaves easily
- Backup *MySQL* without adding load to the server

[9] *Percona XtraBackup* 2.4 only supports *Percona XtraDB Cluster* 5.7.

[1] *InnoDB* tables are still locked while copying non-*InnoDB* data.

[2] Fast incremental backups are supported for *Percona Server for MySQL* with XtraDB changed page tracking enabled.

[8] Backup locks is a lightweight alternative to `FLUSH TABLES WITH READ LOCK` available in *Percona Server for MySQL* 5.6+. *Percona XtraBackup* uses them automatically to copy non-InnoDB data to avoid blocking DML queries that modify *InnoDB* tables.

[3] *Percona XtraBackup* supports encryption with any kinds of backups. *MySQL Enterprise Backup* only supports encryption for single-file backups.

[4] *Percona XtraBackup* performs throttling based on the number of IO operations per second. *MySQL Enterprise Backup* supports a configurable sleep time between operations.

[5] *Percona XtraBackup* skips secondary index pages and recreates them when a compact backup is prepared. *MySQL Enterprise Backup* skips unused pages and reinserts on the prepare stage.

[6] *Percona XtraBackup* can export individual tables even from a full backup, regardless of the InnoDB version. *MySQL Enterprise Backup* uses InnoDB 5.6 transportable tablespaces only when performing a partial backup.

[7] Tables exported with *Percona XtraBackup* can be imported into *Percona Server for MySQL* 5.1, 5.5 or 5.6+, or *MySQL* 5.6+. Transportable tablespaces created with *MySQL Enterprise Backup* can only be imported to *Percona Server for MySQL* 5.6+, *MySQL* 5.6+ or *MariaDB* 10.0+.

Important: *Percona XtraBackup* is not compatible with *MariaDB* 10.3 and later.

CHAPTER

TWO

HOW *PERCONA XTRABACKUP* WORKS

Percona XtraBackup is based on *InnoDB*'s crash-recovery functionality. It copies your *InnoDB* data files, which results in data that is internally inconsistent; but then it performs crash recovery on the files to make them a consistent, usable database again.

This works because *InnoDB* maintains a redo log, also called the transaction log. This contains a record of every change to InnoDB data. When *InnoDB* starts, it inspects the data files and the transaction log, and performs two steps. It applies committed transaction log entries to the data files, and it performs an undo operation on any transactions that modified data but did not commit.

Percona XtraBackup works by remembering the log sequence number (*LSN*) when it starts, and then copying away the data files. It takes some time to do this, so if the files are changing, then they reflect the state of the database at different points in time. At the same time, *Percona XtraBackup* runs a background process that watches the transaction log files, and copies changes from it. *Percona XtraBackup* needs to do this continually because the transaction logs are written in a round-robin fashion, and can be reused after a while. *Percona XtraBackup* needs the transaction log records for every change to the data files since it began execution.

Percona XtraBackup will use Backup locks where available as a lightweight alternative to `FLUSH TABLES WITH READ LOCK`. This feature is available in *Percona Server for MySQL* 5.6+. *Percona XtraBackup* uses this automatically to copy non-InnoDB data to avoid blocking DML queries that modify *InnoDB* tables. When backup locks are supported by the server, **xtrabackup** will first copy *InnoDB* data, run the `LOCK TABLES FOR BACKUP` and copy the *MyISAM* tables and *.frm* files. Once this is done, the backup of the files will begin. It will backup *.frm*, *.MRG*, *.MYD*, *.MYI*, *.TRG*, *.TRN*, *.ARM*, *.ARZ*, *.CSM*, *.CSV*, `.par`, and *.opt* files.

Note: Locking is done only for *MyISAM* and other non-InnoDB tables, and only **after** *Percona XtraBackup* is finished backing up all InnoDB/XtraDB data and logs. *Percona XtraBackup* will use Backup locks where available as a lightweight alternative to `FLUSH TABLES WITH READ LOCK`. This feature is available in *Percona Server for MySQL* 5.6+. *Percona XtraBackup* uses this automatically to copy non-InnoDB data to avoid blocking DML queries that modify *InnoDB* tables.

After that **xtrabackup** will use `LOCK BINLOG FOR BACKUP` to block all operations that might change either binary log position or `Exec_Master_Log_Pos` or `Exec_Gtid_Set` (i.e. master binary log coordinates corresponding to the current SQL thread state on a replication slave) as reported by `SHOW MASTER/SLAVE STATUS`. **xtrabackup** will then finish copying the REDO log files and fetch the binary log coordinates. After this is completed **xtrabackup** will unlock the binary log and tables.

Finally, the binary log position will be printed to `STDERR` and **xtrabackup** will exit returning 0 if all went OK.

Note that the `STDERR` of **xtrabackup** is not written in any file. You will have to redirect it to a file, e.g., `xtrabackup OPTIONS 2> backupout.log`.

It will also create the *following files* in the directory of the backup.

During the prepare phase, *Percona XtraBackup* performs crash recovery against the copied data files, using the copied transaction log file. After this is done, the database is ready to restore and use.

The backed-up *MyISAM* and *InnoDB* tables will be eventually consistent with each other, because after the prepare (recovery) process, *InnoDB*'s data is rolled forward to the point at which the backup completed, not rolled back to the point at which it started. This point in time matches where the `FLUSH TABLES WITH READ LOCK` was taken, so the *MyISAM* data and the prepared *InnoDB* data are in sync.

The **xtrabackup** and **innobackupex** tools both offer many features not mentioned in the preceding explanation. Each tool's functionality is explained in more detail further in the manual. In brief, though, the tools permit you to do operations such as streaming and incremental backups with various combinations of copying the data files, copying the log files, and applying the logs to the data.

Restoring a backup

To restore a backup with **xtrabackup** you can use the *xtrabackup --copy-back* or *xtrabackup --move-back* options.

xtrabackup will read from the `my.cnf` the variables *datadir*, *innodb_data_home_dir*, *innodb_data_file_path*, *innodb_log_group_home_dir* and check that the directories exist.

It will copy the *MyISAM* tables, indexes, etc. (*.frm*, *.MRG*, *.MYD*, *.MYI*, *.TRG*, *.TRN*, *.ARM*, *.ARZ*, *.CSM*, *.CSV*, `par` and *.opt* files) first, *InnoDB* tables and indexes next and the log files at last. It will preserve file's attributes when copying them, you may have to change the files' ownership to `mysql` before starting the database server, as they will be owned by the user who created the backup.

Alternatively, the *xtrabackup --move-back* option may be used to restore a backup. This option is similar to *xtrabackup --copy-back* with the only difference that instead of copying files it moves them to their target locations. As this option removes backup files, it must be used with caution. It is useful in cases when there is not enough free disk space to hold both data files and their backup copies.

Part II

Installation

CHAPTER

THREE

INSTALLING *PERCONA XTRABACKUP* 2.4

This page provides the information on how to install *Percona XtraBackup*. Following options are available:

- *Installing Percona XtraBackup from Repositories* (recommended)
- Installing *Percona XtraBackup* from Downloaded *rpm* or *apt* packages
- *Compiling and Installing from Source Code*

Before installing, you might want to read the *Percona XtraBackup 2.4 Release Notes*.

Installing *Percona XtraBackup* from Repositories

Percona provides repositories for **yum** (`RPM` packages for *Red Hat*, *CentOS* and *Amazon Linux AMI*) and **apt** (`.deb` packages for *Ubuntu* and *Debian*) for software such as *Percona Server for MySQL*, *Percona XtraBackup*, and *Percona Toolkit*. This makes it easy to install and update your software and its dependencies through your operating system's package manager. This is the recommend way of installing where possible.

Following guides describe the installation process for using the official Percona repositories for `.deb` and `.rpm` packages.

Installing *Percona XtraBackup* on *Debian* and *Ubuntu*

Ready-to-use packages are available from the *Percona XtraBackup* software repositories and the download page.

Supported Releases:

- Debian:
- 7.0 (wheezy)
- 8.0 (jessie)
- 9.0 (stretch)
- Ubuntu:
- 14.04LTS (trusty)
- 16.04LTS (xenial)
- 17.04 (zesty)
- 17.10 (artful)
- 18.04 (bionic)

Supported Platforms:

- x86
- x86_64 (also known as `amd64`)

What's in each DEB package?

The `percona-xtrabackup-24` package contains the latest *Percona XtraBackup* GA binaries and associated files.

The `percona-xtrabackup-dbg-24` package contains the debug symbols for binaries in `percona-xtrabackup-24`.

The `percona-xtrabackup-test-24` package contains the test suite for *Percona XtraBackup*.

The `percona-xtrabackup` package contains the older version of the *Percona XtraBackup*.

Installing *Percona XtraBackup* from Percona `apt` repository

1. Fetch the repository packages from Percona web:

```
$ wget https://repo.percona.com/apt/percona-release_latest.$(lsb_release -sc)_all.
↪deb
```

2. Install the downloaded package with **dpkg**. To do that, run the following commands as root or with **sudo**:

```
$ sudo dpkg -i percona-release_latest.$(lsb_release -sc)_all.deb
```

   Once you install this package the Percona repositories should be added. You can check the repository setup in the `/etc/apt/sources.list.d/percona-release.list` file.

3. Remember to update the local cache:

```
$ sudo apt-get update
```

4. After that you can install the package:

```
$ sudo apt-get install percona-xtrabackup-24
```

Percona `apt` Testing repository

Percona offers pre-release builds from the testing repository. To enable it add the just add the `testing` word at the end of the Percona repository definition in your repository file (default `/etc/apt/sources.list.d/percona-release.list`). It should looks like this (in this example `VERSION` is the name of your distribution):

```
deb http://repo.percona.com/apt VERSION main testing
deb-src http://repo.percona.com/apt VERSION main testing
```

For example, if you are running *Debian* 8 (*jessie*) and want to install the latest testing builds, the definitions should look like this:

```
deb http://repo.percona.com/apt jessie main testing
deb-src http://repo.percona.com/apt jessie main testing
```

Apt-Pinning the packages

In some cases you might need to "pin" the selected packages to avoid the upgrades from the distribution repositories. You'll need to make a new file `/etc/apt/preferences.d/00percona.pref` and add the following lines in it:

```
Package: *
Pin: release o=Percona Development Team
Pin-Priority: 1001
```

For more information about the pinning you can check the official debian wiki.

Installing *Percona XtraBackup* using downloaded deb packages

Download the packages of the desired series for your architecture from the download page. Following example will download *Percona XtraBackup* 2.4.4 release package for *Debian* 8.0:

```
$ wget https://www.percona.com/downloads/XtraBackup/Percona-XtraBackup-2.4.4/\
binary/debian/jessie/x86_64/percona-xtrabackup-24_2.4.4-1.jessie_amd64.deb
```

Now you can install *Percona XtraBackup* by running:

```
$ sudo dpkg -i percona-xtrabackup-24_2.4.4-1.jessie_amd64.deb
```

Note: When installing packages manually like this, you'll need to make sure to resolve all the dependencies and install missing packages yourself.

Uninstalling *Percona XtraBackup*

To uninstall *Percona XtraBackup* you'll need to remove all the installed packages.

2. Remove the packages

```
$ sudo apt-get remove percona-xtrabackup-24
```

Installing *Percona XtraBackup* on Red Hat Enterprise Linux and CentOS

Ready-to-use packages are available from the *Percona XtraBackup* software repositories and the download page. The *Percona* **yum** repository supports popular *RPM*-based operating systems, including the *Amazon Linux AMI*.

The easiest way to install the *Percona Yum* repository is to install an *RPM* that configures **yum** and installs the Percona GPG key.

Supported Releases:

- *CentOS* 5 and *RHEL* 5
- *CentOS* 6 and *RHEL* 6 (Current Stable)[1]

[1] "Current Stable": We support only the current stable RHEL6/CentOS6 release, because there is no official (i.e. RedHat provided) method to support or download the latest OpenSSL on RHEL/CentOS versions prior to 6.5. Similarly, and also as a result thereof, there is no official Percona way to support the latest Percona XtraBackup builds on RHEL/CentOS versions prior to 6.5. Additionally, many users will need to upgrade to OpenSSL 1.0.1g or later (due to the Heartbleed vulnerability), and this OpenSSL version is not available for download from any official RHEL/CentOS repository for versions 6.4 and prior. For any officially unsupported system, src.rpm packages may be used to rebuild *Percona XtraBackup* for any environment. Please contact our support service if you require further information on this.

- *CentOS* 7 and *RHEL* 7
- *Amazon Linux AMI* (works the same as *CentOS* 6)

The *CentOS* repositories should work well with *Red Hat Enterprise Linux* too, provided that **yum** is installed on the server.

Supported Platforms:

- x86
- x86_64 (also known as `amd64`)

What's in each RPM package?

The `percona-xtrabackup-24` package contains the latest *Percona XtraBackup* GA binaries and associated files.

The `percona-xtrabackup-24-debuginfo` package contains the debug symbols for binaries in `percona-xtrabackup-24`.

The `percona-xtrabackup-test-24` package contains the test suite for *Percona XtraBackup*.

The `percona-xtrabackup` package contains the older version of the *Percona XtraBackup*.

Installing *Percona XtraBackup* from Percona `yum` repository

1. Install the Percona repository

   You can install Percona yum repository by running the following command as a `root` user or with **sudo**:

   ```
   $ yum install https://repo.percona.com/yum/percona-release-latest.noarch.rpm
   ```

   You should see some output such as the following:

   ```
   Retrieving https://repo.percona.com/yum/percona-release-latest.noarch.rpm
   Preparing...                ########################################### [100%]
      1:percona-release        ########################################### [100%]
   ```

Note: *RHEL*/*Centos* 5 doesn't support installing the packages directly from the remote location so you'll need to download the package first and install it manually with **rpm**:

```
$ wget https://repo.percona.com/yum/percona-release-latest.noarch.rpm
$ rpm -ivH percona-release-latest.noarch.rpm
```

2. Testing the repository

   Make sure packages are now available from the repository, by executing the following command:

   ```
   yum list | grep percona
   ```

   You should see output similar to the following:

   ```
   ...
   percona-xtrabackup-20.x86_64               2.0.8-587.rhel5             percona-
    ↪release-x86_64
   percona-xtrabackup-20-debuginfo.x86_64     2.0.8-587.rhel5             percona-
    ↪release-x86_64
   ```

```
percona-xtrabackup-20-test.x86_64          2.0.8-587.rhel5            percona-
↪release-x86_64
percona-xtrabackup-21.x86_64               2.1.9-746.rhel5            percona-
↪release-x86_64
percona-xtrabackup-21-debuginfo.x86_64     2.1.9-746.rhel5            percona-
↪release-x86_64
percona-xtrabackup-22.x86_64               2.2.13-1.el5               percona-
↪release-x86_64
percona-xtrabackup-22-debuginfo.x86_64     2.2.13-1.el5               percona-
↪release-x86_64
percona-xtrabackup-debuginfo.x86_64        2.3.5-1.el5                percona-
↪release-x86_64
percona-xtrabackup-test.x86_64             2.3.5-1.el5                percona-
↪release-x86_64
percona-xtrabackup-test-21.x86_64          2.1.9-746.rhel5            percona-
↪release-x86_64
percona-xtrabackup-test-22.x86_64          2.2.13-1.el5               percona-
↪release-x86_64
...
```

3. Install the packages

   You can now install *Percona XtraBackup* by running:

```
yum install percona-xtrabackup-24
```

Warning: In order to sucessfully install *Percona XtraBackup* on CentOS prior to version 7, the `libev` package needs to be installed first. This package `libev` package can be installed from the EPEL repositories.

Percona *yum* Testing Repository

Percona offers pre-release builds from our testing repository. To subscribe to the testing repository, you'll need to enable the testing repository in `/etc/yum.repos.d/percona-release.repo`. To do so, set both `percona-testing-$basearch` and `percona-testing-noarch` to `enabled = 1` (Note that there are 3 sections in this file: release, testing and experimental - in this case it is the second section that requires updating). **NOTE:** You'll need to install the Percona repository first (ref above) if this hasn't been done already.

Installing *Percona XtraBackup* using downloaded rpm packages

Download the packages of the desired series for your architecture from the download page. Following example will download *Percona XtraBackup* 2.4.4 release package for *CentOS* 7:

```
$ wget https://www.percona.com/downloads/XtraBackup/Percona-XtraBackup-2.4.4/\
binary/redhat/7/x86_64/percona-xtrabackup-24-2.4.4-1.el7.x86_64.rpm
```

Now you can install *Percona XtraBackup* by running:

```
$ yum localinstall percona-xtrabackup-24-2.4.4-1.el7.x86_64.rpm
```

Note: When installing packages manually like this, you'll need to make sure to resolve all the dependencies and install missing packages yourself.

Uninstalling *Percona XtraBackup*

To completely uninstall *Percona XtraBackup* you'll need to remove all the installed packages.

Remove the packages

```
yum remove percona-xtrabackup
```

Compiling and Installing from Source Code

Percona XtraBackup is open source and the code is available on Github. Following guide describes the compiling and installation process from source code.

Compiling and Installing from Source Code

The source code is available from the *Percona XtraBackup Github* project. The easiest way to get the code is with **`git clone`** and switch to the desired release branch, such as the following:

```
$ git clone https://github.com/percona/percona-xtrabackup.git
$ cd percona-xtrabackup
$ git checkout 2.4
```

You should then have a directory named after the release you branched, such as `percona-xtrabackup`.

Compiling on Linux

Prerequisites

The following packages and tools must be installed to compile *Percona XtraBackup* from source. These might vary from system to system.

In Debian-based distributions, you need to:

```
$ apt-get install build-essential flex bison automake autoconf \
   libtool cmake libaio-dev mysql-client libncurses-dev zlib1g-dev \
   libgcrypt11-dev libev-dev libcurl4-gnutls-dev vim-common
```

In `RPM`-based distributions, you need to:

```
$ yum install cmake gcc gcc-c++ libaio libaio-devel automake autoconf \
  bison libtool ncurses-devel libgcrypt-devel libev-devel libcurl-devel \
  vim-common
```

Compiling with CMake

At the base directory of the source code tree, if you execute:

```
$ cmake -DBUILD_CONFIG=xtrabackup_release -DWITH_MAN_PAGES=OFF && make -j4
```

and you go for a coffee, at your return *Percona XtraBackup* will be ready to be used.

Note: You can build *Percona XtraBackup* with man pages but this requires `python-sphinx` package which isn't available from that main repositories for every distribution. If you installed the `python-sphinx` package you need to remove the `-DWITH_MAN_PAGES=OFF` from previous command.

Installation

The following command:

```
$ make install
```

will install all *Percona XtraBackup* binaries, the **innobackupex** script and tests to `/usr/local/xtrabackup`. You can override this either with:

```
$ make DESTDIR=... install
```

or by changing the installation layout with:

```
$ cmake -DINSTALL_LAYOUT=...
```

Part III

Prerequisites

CHAPTER

FOUR

CONNECTION AND PRIVILEGES NEEDED

Percona XtraBackup needs to be able to connect to the database server and perform operations on the server and the *datadir* when creating a backup, when preparing in some scenarios and when restoring it. In order to do so, there are privileges and permission requirements on its execution that must be fulfilled.

Privileges refers to the operations that a system user is permitted to do in the database server. **They are set at the database server and only apply to users in the database server**.

Permissions are those which permits a user to perform operations on the system, like reading, writing or executing on a certain directory or start/stop a system service. **They are set at a system level and only apply to system users**.

Whether **xtrabackup** or **innobackupex** is used, there are two actors involved: the user invoking the program - *a system user* - and the user performing action in the database server - *a database user*. Note that these are different users in different places, even though they may have the same username.

All the invocations of **innobackupex** and **xtrabackup** in this documentation assume that the system user has the appropriate permissions and you are providing the relevant options for connecting the database server - besides the options for the action to be performed - and the database user has adequate privileges.

Connecting to the server

The database user used to connect to the server and its password are specified by the *xtrabackup --user* and *xtrabackup --password* option:

```
$ xtrabackup --user=DVADER --password=14MY0URF4TH3R --backup \
  --target-dir=/data/bkps/
$ innobackupex --user=DBUSER --password=SECRET /path/to/backup/dir/
$ innobackupex --user=LUKE  --password=US3TH3F0RC3 --stream=tar ./ | bzip2 -
```

If you don't use the *xtrabackup --user* option, *Percona XtraBackup* will assume the database user whose name is the system user executing it.

Other Connection Options

According to your system, you may need to specify one or more of the following options to connect to the server:

Option	Description
–port	The port to use when connecting to the database server with TCP/IP.
–socket	The socket to use when connecting to the local database.
–host	The host to use when connecting to the database server with TCP/IP.

These options are passed to the **mysql** child process without alteration, see `mysql --help` for details.

Note: In case of multiple server instances the correct connection parameters (port, socket, host) must be specified in order for **xtrabackup** to talk to the correct server.

Permissions and Privileges Needed

Once connected to the server, in order to perform a backup you will need `READ` and `EXECUTE` permissions at a filesystem level in the server's *datadir*.

The database user needs the following privileges on the tables/databases to be backed up:

- `RELOAD` and `LOCK TABLES` (unless the *`--no-lock`* option is specified) in order to `FLUSH TABLES WITH READ LOCK` and `FLUSH ENGINE LOGS` prior to start copying the files, and `LOCK TABLES FOR BACKUP` and `LOCK BINLOG FOR BACKUP` require this privilege when Backup Locks are used,
- `REPLICATION CLIENT` in order to obtain the binary log position,
- `CREATE TABLESPACE` in order to import tables (see *Restoring Individual Tables*),
- `PROCESS` in order to run `SHOW ENGINE INNODB STATUS` (which is mandatory), and optionally to see all threads which are running on the server (see *Improved FLUSH TABLES WITH READ LOCK handling*),
- `SUPER` in order to start/stop the slave threads in a replication environment, use XtraDB Changed Page Tracking for *Incremental Backups* and for *Improved FLUSH TABLES WITH READ LOCK handling*,
- `CREATE` privilege in order to create the *PERCONA_SCHEMA.xtrabackup_history* database and table,
- `INSERT` privilege in order to add history records to the *PERCONA_SCHEMA.xtrabackup_history* table,
- `SELECT` privilege in order to use *`innobackupex --incremental-history-name`* or *`innobackupex --incremental-history-uuid`* in order for the feature to look up the `innodb_to_lsn` values in the *PERCONA_SCHEMA.xtrabackup_history* table.

The explanation of when these are used can be found in *How Percona XtraBackup Works*.

An SQL example of creating a database user with the minimum privileges required to full backups would be:

```
mysql> CREATE USER 'bkpuser'@'localhost' IDENTIFIED BY 's3cret';
mysql> GRANT RELOAD, LOCK TABLES, PROCESS, REPLICATION CLIENT ON *.* TO
       'bkpuser'@'localhost';
mysql> FLUSH PRIVILEGES;
```

CHAPTER

FIVE

CONFIGURING XTRABACKUP

All of the **xtrabackup** configuration is done through options, which behave exactly like standard *MySQL* program options: they can be specified either at the command-line, or through a file such as `/etc/my.cnf`.

The **xtrabackup** binary reads the `[mysqld]` and `[xtrabackup]` sections from any configuration files, in that order. That is so that it can read its options from your existing *MySQL* installation, such as the *datadir* or some of the *InnoDB* options. If you want to override these, just specify them in the `[xtrabackup]` section, and because it is read later, it will take precedence.

You don't need to put any configuration in your `my.cnf` if you don't want to. You can simply specify the options on the command-line. Normally, the only thing you might find convenient to place in the `[xtrabackup]` section of your `my.cnf` file is the `target_dir` option to default the directory in which the backups will be placed, for example:

```
[xtrabackup]
target_dir = /data/backups/mysql/
```

This manual will assume that you do not have any file-based configuration for **xtrabackup**, so it will always show command-line options being used explicitly. Please see the *option and variable reference* for details on all of the configuration options.

The **xtrabackup** binary does not accept exactly the same syntax in the `my.cnf` file as the **mysqld** server binary does. For historical reasons, the **mysqld** server binary accepts parameters with a `--set-variable=<variable>=<value>` syntax, which **xtrabackup** does not understand. If your `my.cnf` file has such configuration directives, you should rewrite them in the `--variable=value` syntax.

System Configuration and NFS Volumes

The **xtrabackup** tool requires no special configuration on most systems. However, the storage where the *`xtrabackup --target-dir`* is located must behave properly when `fsync()` is called. In particular, we have noticed that NFS volumes not mounted with the `sync` option might not really sync the data. As a result, if you back up to an NFS volume mounted with the async option, and then try to prepare the backup from a different server that also mounts that volume, the data might appear to be corrupt. You can use the `sync` mount option to avoid this problem.

Part IV

Backup Scenarios

CHAPTER

SIX

THE BACKUP CYCLE - FULL BACKUPS

Creating a backup

To create a backup, run **xtrabackup** with the *xtrabackup --backup* option. You also need to specify a *xtrabackup --target-dir* option, which is where the backup will be stored, if the *InnoDB* data or log files aren't stored in the same directory, you might need to specify the location of those, too. If the target directory does not exist, **xtrabackup** creates it. If the directory does exist and is empty, **xtrabackup** will succeed. **xtrabackup** will not overwrite existing files, it will fail with operating system error 17, `file exists`.

To start the backup process run:

```
$ xtrabackup --backup --target-dir=/data/backups/
```

This will store the backup at `/data/backups/`. If you specify a relative path, the target directory will be relative to the current directory.

During the backup process, you should see a lot of output showing the data files being copied, as well as the log file thread repeatedly scanning the log files and copying from it. Here is an example that shows the log thread scanning the log in the background, and a file copying thread working on the `ibdata1` file:

```
160906 10:19:17 Finished backing up non-InnoDB tables and files
160906 10:19:17 Executing FLUSH NO_WRITE_TO_BINLOG ENGINE LOGS...
xtrabackup: The latest check point (for incremental): '62988944'
xtrabackup: Stopping log copying thread.
.160906 10:19:18 >> log scanned up to (137343534)
160906 10:19:18 Executing UNLOCK TABLES
160906 10:19:18 All tables unlocked
160906 10:19:18 Backup created in directory '/data/backups/'
160906 10:19:18 [00] Writing backup-my.cnf
160906 10:19:18 [00]        ...done
160906 10:19:18 [00] Writing xtrabackup_info
160906 10:19:18 [00]        ...done
xtrabackup: Transaction log of lsn (26970807) to (137343534) was copied.
160906 10:19:18 completed OK!
```

The last thing you should see is something like the following, where the value of the `<LSN>` will be a number that depends on your system:

```
xtrabackup: Transaction log of lsn (<SLN>) to (<LSN>) was copied.
```

Note: Log copying thread checks the transactional log every second to see if there were any new log records written that need to be copied, but there is a chance that the log copying thread might not be able to keep up with the amount

of writes that go to the transactional logs, and will hit an error when the log records are overwritten before they could be read.

After the backup is finished, the target directory will contain files such as the following, assuming you have a single InnoDB table `test.tbl1` and you are using MySQL's *innodb_file_per_table* option:

```
$ ls -lh /data/backups/
total 182M
drwx------  7 root root 4.0K Sep  6 10:19 .
drwxrwxrwt 11 root root 4.0K Sep  6 11:05 ..
-rw-r-----  1 root root  387 Sep  6 10:19 backup-my.cnf
-rw-r-----  1 root root  76M Sep  6 10:19 ibdata1
drwx------  2 root root 4.0K Sep  6 10:19 mysql
drwx------  2 root root 4.0K Sep  6 10:19 performance_schema
drwx------  2 root root 4.0K Sep  6 10:19 sbtest
drwx------  2 root root 4.0K Sep  6 10:19 test
drwx------  2 root root 4.0K Sep  6 10:19 world2
-rw-r-----  1 root root  116 Sep  6 10:19 xtrabackup_checkpoints
-rw-r-----  1 root root  433 Sep  6 10:19 xtrabackup_info
-rw-r-----  1 root root 106M Sep  6 10:19 xtrabackup_logfile
```

The backup can take a long time, depending on how large the database is. It is safe to cancel at any time, because it does not modify the database.

The next step is getting your backup ready to be restored.

Preparing a backup

After you made a backup with the *xtrabackup --backup* option, you'll first need to prepare it in order to restore it. Data files are not point-in-time consistent until they've been prepared, because they were copied at different times as the program ran, and they might have been changed while this was happening. If you try to start InnoDB with these data files, it will detect corruption and crash itself to prevent you from running on damaged data. The *xtrabackup --prepare* step makes the files perfectly consistent at a single instant in time, so you can run *InnoDB* on them.

You can run the prepare operation on any machine; it does not need to be on the originating server or the server to which you intend to restore. You can copy the backup to a utility server and prepare it there.

Note: You can prepare a backup created with older *Percona XtraBackup* version with a newer one, but not vice versa. Preparing a backup on an unsupported server version should be done with the latest *Percona XtraBackup* release which supports that server version. For example, if one has a backup of MySQL 5.0 created with *Percona XtraBackup* 1.6, then preparing the backup with *Percona XtraBackup* 2.3 is not supported, because support for MySQL 5.0 was removed in *Percona XtraBackup* 2.1. Instead, the latest release in the 2.0 series should be used.

During the `prepare` operation, **`xtrabackup`** boots up a kind of modified InnoDB that's embedded inside it (the libraries it was linked against). The modifications are necessary to disable InnoDB's standard safety checks, such as complaining that the log file isn't the right size, which aren't appropriate for working with backups. These modifications are only for the xtrabackup binary; you don't need a modified *InnoDB* to use **`xtrabackup`** for your backups.

The prepare step uses this *embedded InnoDB* to perform crash recovery on the copied data files, using the copied log file. The `prepare` step is very simple to use: you simply run *xtrabackup --prepare* option and tell it which directory to prepare, for example, to prepare the previously taken backup run:

```
$ xtrabackup --prepare --target-dir=/data/backups/
```

When this finishes, you should see an `InnoDB shutdown` with a message such as the following, where again the value of *LSN* will depend on your system:

```
InnoDB: Shutdown completed; log sequence number 137345046
160906 11:21:01 completed OK!
```

All following prepares will not change the already prepared data files, you'll see that output says:

```
xtrabackup: This target seems to be already prepared.
xtrabackup: notice: xtrabackup_logfile was already used to '--prepare'.
```

It is not recommended to interrupt xtrabackup process while preparing backup because it may cause data files corruption and backup will become unusable. Backup validity is not guaranteed if prepare process was interrupted.

Note: If you intend the backup to be the basis for further incremental backups, you should use the *`xtrabackup --apply-log-only`* option when preparing the backup, or you will not be able to apply incremental backups to it. See the documentation on preparing *incremental backups* for more details.

Restoring a Backup

Warning: Backup needs to be *prepared* before it can be restored.

For convenience **`xtrabackup`** binary has an *`xtrabackup --copy-back`* option, which will copy the backup to the server's *datadir*:

```
$ xtrabackup --copy-back --target-dir=/data/backups/
```

If you don't want to save your backup, you can use the *`xtrabackup --move-back`* option which will move the backed up data to the *datadir*.

If you don't want to use any of the above options, you can additionally use **`rsync`** or **`cp`** to restore the files.

Note: The *datadir* must be empty before restoring the backup. Also it's important to note that MySQL server needs to be shut down before restore is performed. You can't restore to a *datadir* of a running mysqld instance (except when importing a partial backup).

Example of the **`rsync`** command that can be used to restore the backup can look like this:

```
$ rsync -avrP /data/backup/ /var/lib/mysql/
```

You should check that the restored files have the correct ownership and permissions.

As files' attributes will be preserved, in most cases you will need to change the files' ownership to `mysql` before starting the database server, as they will be owned by the user who created the backup:

```
$ chown -R mysql:mysql /var/lib/mysql
```

Data is now restored and you can start the server.

CHAPTER

SEVEN

INCREMENTAL BACKUP

Both **xtrabackup** and **innobackupex** tools supports incremental backups, which means that they can copy only the data that has changed since the last backup.

You can perform many incremental backups between each full backup, so you can set up a backup process such as a full backup once a week and an incremental backup every day, or full backups every day and incremental backups every hour.

Incremental backups work because each *InnoDB* page contains a log sequence number, or *LSN*. The *LSN* is the system version number for the entire database. Each page's *LSN* shows how recently it was changed.

An incremental backup copies each page whose *LSN* is newer than the previous incremental or full backup's *LSN*. There are two algorithms in use to find the set of such pages to be copied. The first one, available with all the server types and versions, is to check the page *LSN* directly by reading all the data pages. The second one, available with *Percona Server for MySQL*, is to enable the changed page tracking feature on the server, which will note the pages as they are being changed. This information will be then written out in a compact separate so-called bitmap file. The **xtrabackup** binary will use that file to read only the data pages it needs for the incremental backup, potentially saving many read requests. The latter algorithm is enabled by default if the **xtrabackup** binary finds the bitmap file. It is possible to specify *xtrabackup --incremental-force-scan* to read all the pages even if the bitmap data is available.

Incremental backups do not actually compare the data files to the previous backup's data files. In fact, you can use *xtrabackup --incremental-lsn* to perform an incremental backup without even having the previous backup, if you know its *LSN*. Incremental backups simply read the pages and compare their *LSN* to the last backup's *LSN*. You still need a full backup to recover the incremental changes, however; without a full backup to act as a base, the incremental backups are useless.

Creating an Incremental Backup

To make an incremental backup, begin with a full backup as usual. The **xtrabackup** binary writes a file called `xtrabackup_checkpoints` into the backup's target directory. This file contains a line showing the `to_lsn`, which is the database's *LSN* at the end of the backup. *Create the full backup* with a following command:

```
$ xtrabackup --backup --target-dir=/data/backups/base
```

If you look at the `xtrabackup_checkpoints` file, you should see similar content depending on your LSN nuber:

```
backup_type = full-backuped
from_lsn = 0
to_lsn = 1626007
last_lsn = 1626007
compact = 0
recover_binlog_info = 1
```

Now that you have a full backup, you can make an incremental backup based on it. Use the following command:

```
$ xtrabackup --backup --target-dir=/data/backups/inc1 \
--incremental-basedir=/data/backups/base
```

The `/data/backups/inc1/` directory should now contain delta files, such as `ibdata1.delta` and `test/table1.ibd.delta`. These represent the changes since the `LSN 1626007`. If you examine the `xtrabackup_checkpoints` file in this directory, you should see similar content to the following:

```
backup_type = incremental
from_lsn = 1626007
to_lsn = 4124244
last_lsn = 4124244
compact = 0
recover_binlog_info = 1
```

`from_lsn` is the starting LSN of the backup and for incremental it has to be the same as `to_lsn` (if it is the last checkpoint) of the previous/base backup.

It's now possible to use this directory as the base for yet another incremental backup:

```
$ xtrabackup --backup --target-dir=/data/backups/inc2 \
--incremental-basedir=/data/backups/inc1
```

This folder also contains the `xtrabackup_checkpoints`:

```
backup_type = incremental
from_lsn = 4124244
to_lsn = 6938371
last_lsn = 7110572
compact = 0
recover_binlog_info = 1
```

Note: In this case you can see that there is a difference between the `to_lsn` (last checkpoint LSN) and `last_lsn` (last copied LSN), this means that there was some traffic on the server during the backup process.

Preparing the Incremental Backups

The *`xtrabackup --prepare`* step for incremental backups is not the same as for full backups. In full backups, two types of operations are performed to make the database consistent: committed transactions are replayed from the log file against the data files, and uncommitted transactions are rolled back. You must skip the rollback of uncommitted transactions when preparing an incremental backup, because transactions that were uncommitted at the time of your backup may be in progress, and it's likely that they will be committed in the next incremental backup. You should use the *`xtrabackup --apply-log-only`* option to prevent the rollback phase.

Warning: If you do not use the *`xtrabackup --apply-log-only`* option to prevent the rollback phase, then your incremental backups will be useless. After transactions have been rolled back, further incremental backups cannot be applied.

Beginning with the full backup you created, you can prepare it, and then apply the incremental differences to it. Recall that you have the following backups:

```
/data/backups/base
/data/backups/inc1
/data/backups/inc2
```

To prepare the base backup, you need to run *xtrabackup --prepare* as usual, but prevent the rollback phase:

```
$ xtrabackup --prepare --apply-log-only --target-dir=/data/backups/base
```

The output should end with text similar to the following:

```
InnoDB: Shutdown completed; log sequence number 1626007
161011 12:41:04 completed OK!
```

The log sequence number should match the `to_lsn` of the base backup, which you saw previously.

Note: This backup is actually safe to *restore* as-is now, even though the rollback phase has been skipped. If you restore it and start *MySQL*, *InnoDB* will detect that the rollback phase was not performed, and it will do that in the background, as it usually does for a crash recovery upon start. It will notify you that the database was not shut down normally.

To apply the first incremental backup to the full backup, run the following command:

```
$ xtrabackup --prepare --apply-log-only --target-dir=/data/backups/base \
--incremental-dir=/data/backups/inc1
```

This applies the delta files to the files in `/data/backups/base`, which rolls them forward in time to the time of the incremental backup. It then applies the redo log as usual to the result. The final data is in `/data/backups/base`, not in the incremental directory. You should see an output similar to:

```
incremental backup from 1626007 is enabled.
xtrabackup: cd to /data/backups/base
xtrabackup: This target seems to be already prepared with --apply-log-only.
xtrabackup: xtrabackup_logfile detected: size=2097152, start_lsn=(4124244)
...
xtrabackup: page size for /tmp/backups/inc1/ibdata1.delta is 16384 bytes
Applying /tmp/backups/inc1/ibdata1.delta to ./ibdata1...
...
161011 12:45:56 completed OK!
```

Again, the *LSN* should match what you saw from your earlier inspection of the first incremental backup. If you restore the files from `/data/backups/base`, you should see the state of the database as of the first incremental backup.

Warning: PXB does not support using the same incremental backup directory to prepare two copies of backup. Do not run *xtrabackup --prepare* with the same incremental backup directory (the value of *–incremental-dir*) more than once.

Preparing the second incremental backup is a similar process: apply the deltas to the (modified) base backup, and you will roll its data forward in time to the point of the second incremental backup:

```
$ xtrabackup --prepare --target-dir=/data/backups/base \
--incremental-dir=/data/backups/inc2
```

Note: *xtrabackup --apply-log-only* should be used when merging all incrementals except the last

one. That's why the previous line doesn't contain the `xtrabackup --apply-log-only` option. Even if the `xtrabackup --apply-log-only` was used on the last step, backup would still be consistent but in that case server would perform the rollback phase.

Once prepared, incremental backups are the same as the *full backups* and they can be *restored* in the same way.

CHAPTER

EIGHT

COMPRESSED BACKUP

Percona XtraBackup has implemented support for compressed backups. It can be used to compress/decompress local or streaming backup with *xbstream*.

Creating Compressed Backups

In order to make a compressed backup you'll need to use *xtrabackup --compress* option:

```
$ xtrabackup --backup --compress --target-dir=/data/compressed/
```

If you want to speed up the compression you can use the parallel compression, which can be enabled with *xtrabackup --compress-threads* option. Following example will use four threads for compression:

```
$ xtrabackup --backup --compress --compress-threads=4 \
--target-dir=/data/compressed/
```

Output should look like this

```
...
170223 13:00:38 [01] Compressing ./test/sbtest1.frm to /tmp/compressed/test/sbtest1.
↪frm.qp
170223 13:00:38 [01]        ...done
170223 13:00:38 [01] Compressing ./test/sbtest2.frm to /tmp/compressed/test/sbtest2.
↪frm.qp
170223 13:00:38 [01]        ...done
...
170223 13:00:39 [00] Compressing xtrabackup_info
170223 13:00:39 [00]        ...done
xtrabackup: Transaction log of lsn (9291934) to (9291934) was copied.
170223 13:00:39 completed OK!
```

Preparing the backup

Before you can prepare the backup you'll need to uncompress all the files. *Percona XtraBackup* has implemented *xtrabackup --decompress* option that can be used to decompress the backup.

Note: Before proceeding you'll need to make sure that qpress has been installed. It's availabe from *Percona Software repositories*

```
$ xtrabackup --decompress --target-dir=/data/compressed/
```

Note: *xtrabackup --parallel* can be used with *xtrabackup --decompress* option to decompress multiple files simultaneously.

Percona XtraBackup doesn't automatically remove the compressed files. In order to clean up the backup directory you should use *xtrabackup --remove-original* option. Even if they're not removed these files will not be copied/moved over to the datadir if *xtrabackup --copy-back* or *xtrabackup --move-back* are used.

When the files are uncompressed you can prepare the backup with the *xtrabackup --prepare* option:

```
$ xtrabackup --prepare --target-dir=/data/compressed/
```

You should check for a confirmation message:

```
InnoDB: Starting shutdown...
InnoDB: Shutdown completed; log sequence number 9293846
170223 13:39:31 completed OK!
```

Now the files in `/data/compressed/` are ready to be used by the server.

Restoring the backup

xtrabackup has a *xtrabackup --copy-back* option, which performs the restoration of a backup to the server's *datadir*:

```
$ xtrabackup --copy-back --target-dir=/data/backups/
```

It will copy all the data-related files back to the server's *datadir*, determined by the server's `my.cnf` configuration file. You should check the last line of the output for a success message:

```
170223 13:49:13 completed OK!
```

You should check the file permissions after copying the data back. You may need to adjust them with something like:

```
$ chown -R mysql:mysql /var/lib/mysql
```

Now that the *datadir* contains the restored data. You are ready to start the server.

CHAPTER

NINE

ENCRYPTED BACKUP

Percona XtraBackup has implemented support for encrypted backups. It can be used to encrypt/decrypt local or streaming backup with *xbstream* option (streaming tar backups are not supported) in order to add another layer of protection to the backups. Encryption is done with the `libgcrypt` library.

Creating Encrypted Backups

To make an encrypted backup following options need to be specified (options *xtrabackup --encrypt-key* and *xtrabackup --encrypt-key-file* are mutually exclusive, i.e., just one of them needs to be provided):

- `--encrypt=ALGORITHM` - currently supported algorithms are: `AES128`, `AES192` and `AES256`
- `--encrypt-key=ENCRYPTION_KEY` - proper length encryption key to use. It is not recommended to use this option where there is uncontrolled access to the machine as the command line and thus the key can be viewed as part of the process info.
- `--encrypt-key-file=KEYFILE` - the name of a file where the raw key of the appropriate length can be read from. The file must be a simple binary (or text) file that contains exactly the key to be used.

Both *xtrabackup --encrypt-key* option and *xtrabackup --encrypt-key-file* option can be used to specify the encryption key. Encryption key can be generated with command like:

```
$ openssl rand -base64 24
```

Example output of that command should look like this:

```
GCHFLrDFVx6UAsRb88uLVbAVWbK+Yzfs
```

This value then can be used as the encryption key

Using the `--encrypt-key` option

Example of the xtrabackup command using the *xtrabackup --encrypt-key* should look like this:

```
$ xtrabackup --backup --target-dir=/data/backups --encrypt=AES256 \
--encrypt-key="GCHFLrDFVx6UAsRb88uLVbAVWbK+Yzfs"
```

Using the `--encrypt-key-file` option

Example of the xtrabackup command using the *xtrabackup --encrypt-key-file* should look like this:

```
$ xtrabackup --backup --target-dir=/data/backups/ --encrypt=AES256 \
--encrypt-key-file=/data/backups/keyfile
```

Note: Depending on the text editor used for making the `KEYFILE`, text file in some cases can contain the CRLF and this will cause the key size to grow and thus making it invalid. Suggested way to do this would be to create the file with: `echo -n "GCHFLrDFVx6UAsRb88uLVbAVWbK+Yzfs" > /data/backups/keyfile`

Optimizing the encryption process

Two options have been introduced with the encrypted backups that can be used to speed up the encryption process. These are *xtrabackup --encrypt-threads* and *xtrabackup --encrypt-chunk-size*. By using the *xtrabackup --encrypt-threads* option multiple threads can be specified to be used for encryption in parallel. Option *xtrabackup --encrypt-chunk-size* can be used to specify the size (in bytes) of the working encryption buffer for each encryption thread (default is 64K).

Decrypting Encrypted Backups

Percona XtraBackup *xtrabackup --decrypt* option has been implemented that can be used to decrypt the backups:

```
$ xtrabackup --decrypt=AES256 --encrypt-key="GCHFLrDFVx6UAsRb88uLVbAVWbK+Yzfs"\
--target-dir=/data/backups/
```

Percona XtraBackup doesn't automatically remove the encrypted files. In order to clean up the backup directory users should remove the `*.xbcrypt` files. In *Percona XtraBackup* 2.4.6 *xtrabackup --remove-original* option has been implemented that you can use to remove the encrypted files once they've been decrypted. To remove the files once they're decrypted you should run:

```
$ xtrabackup --decrypt=AES256 --encrypt-key="GCHFLrDFVx6UAsRb88uLVbAVWbK+Yzfs"\
--target-dir=/data/backups/ --remove-original
```

Note: *xtrabackup --parallel* can be used with *xtrabackup --decrypt* option to decrypt multiple files simultaneously.

When the files have been decrypted backup can be prepared.

Preparing Encrypted Backups

After the backups have been decrypted, they can be prepared the same way as the standard full backups with the *xtrabackup --prepare* option:

```
$ xtrabackup --prepare --target-dir=/data/backups/
```

Restoring Encrypted Backups

xtrabackup has a *xtrabackup --copy-back* option, which performs the restoration of a backup to the server's *datadir*:

```
$ xtrabackup --copy-back --target-dir=/data/backups/
```

It will copy all the data-related files back to the server's *datadir*, determined by the server's `my.cnf` configuration file. You should check the last line of the output for a success message:

```
170214 12:37:01 completed OK!
```

Other Reading

- The Libgcrypt Reference Manual

Part V

User's Manual

CHAPTER

TEN

PERCONA XTRABACKUP USER MANUAL

The innobackupex Program

The **innobackupex** program is a symlink to the *xtrabackup C* program. It lets you perform point-in-time backups of *InnoDB* / *XtraDB* tables together with the schema definitions, *MyISAM* tables, and other portions of the server. In previous versions **innobackupex** was implemented as a *Perl* script.

This manual section explains how to use **innobackupex** in detail.

Warning: The **innobackupex** program is deprecated. Please switch to **xtrabackup**.

The Backup Cycle - Full Backups

Creating a Backup with `innobackupex`

innobackupex is the tool which provides functionality to backup a whole MySQL database instance using the **xtrabackup** in combination with tools like *xbstream* and *xbcrypt*.

To create a full backup, invoke the script with the options needed to connect to the server and only one argument: the path to the directory where the backup will be stored

```
$ innobackupex --user=DBUSER --password=DBUSERPASS /path/to/BACKUP-DIR/
```

and check the last line of the output for a confirmation message:

```
innobackupex: Backup created in directory '/path/to/BACKUP-DIR/2013-03-25_00-00-09'
innobackupex: MySQL binlog position: filename 'mysql-bin.000003', position 1946
111225 00:00:53  innobackupex: completed OK!
```

The backup will be stored within a time stamped directory created in the provided path, `/path/to/BACKUP-DIR/2013-03-25_00-00-09` in this particular example.

Under the hood

innobackupex called **xtrabackup** binary to backup all the data of *InnoDB* tables (see *Creating a backup* for details on this process) and copied all the table definitions in the database (*.frm* files), data and files related to *MyISAM*, *MERGE* (reference to other tables), *CSV* and *ARCHIVE* tables, along with *triggers* and *database configuration information* to a time stamped directory created in the provided path.

It will also create the *following files* for convenience on the created directory.

innobackupex --no-timestamp

This option tells **innobackupex** not to create a time stamped directory to store the backup:

```
$ innobackupex --user=DBUSER --password=DBUSERPASS /path/to/BACKUP-DIR/ --no-timestamp
```

innobackupex will create the `BACKUP-DIR` subdirectory (or fail if exists) and store the backup inside of it.

innobackupex --defaults-file

You can provide another configuration file to **innobackupex** with this option. The only limitation is that **it has to be the first option passed**:

```
$ innobackupex --defaults-file=/tmp/other-my.cnf --user=DBUSER --password=DBUSERPASS /
↪path/to/BACKUP-DIR/
```

Preparing a Full Backup with innobackupex

After creating a backup, the data is not ready to be restored. There might be uncommitted transactions to be undone or transactions in the logs to be replayed. Doing those pending operations will make the data files consistent and it is the purpose of the **prepare stage**. Once this has been done, the data is ready to be used.

To prepare a backup with **innobackupex** you have to use the *innobackupex --apply-log* and full path to the backup directory as an argument:

```
$ innobackupex --apply-log /path/to/BACKUP-DIR
```

and check the last line of the output for a confirmation on the process:

```
150806 01:01:57  InnoDB: Shutdown completed; log sequence number 1609228
150806 01:01:57  innobackupex: completed OK!
```

If it succeeded, **innobackupex** performed all operations needed, leaving the data ready to use immediately.

Under the hood

innobackupex started the prepare process by reading the configuration from the `backup-my.cnf` file in the backup directory.

After that, **innobackupex** replayed the committed transactions in the log files (some transactions could have been done while the backup was being done) and rolled back the uncommitted ones. Once this is done, all the information lay in the tablespace (the InnoDB files), and the log files are re-created.

This implies calling *innobackupex --apply-log* twice. More details of this process are shown in the *xtrabackup section*.

Note that this preparation is not suited for incremental backups. If you perform it on the base of an incremental backup, you will not be able to "add" the increments. See *Incremental Backups with innobackupex*.

Other options to consider

innobackupex --use-memory

The preparing process can be sped up by using more memory in it. It depends on the free or available RAM on your system, it defaults to 100MB. In general, the more memory available to the process, the better. The amount of memory used in the process can be specified by multiples of bytes:

```
$ innobackupex --apply-log --use-memory=4G /path/to/BACKUP-DIR
```

Restoring a Full Backup with innobackupex

For convenience, **innobackupex** has a *innobackupex --copy-back* option, which performs the restoration of a backup to the server's *datadir*:

```
$ innobackupex --copy-back /path/to/BACKUP-DIR
```

It will copy all the data-related files back to the server's *datadir*, determined by the server's my.cnf configuration file. You should check the last line of the output for a success message:

```
innobackupex: Finished copying back files.

111225 01:08:13  innobackupex: completed OK!
```

Note: The *datadir* must be empty; *Percona XtraBackup* *innobackupex --copy-back* option will not copy over existing files unless *innobackupex --force-non-empty-directories* option is specified. Also it is important to note that *MySQL* server needs to be shut down before restore is performed. You can't restore to a *datadir* of a running mysqld instance (except when importing a partial backup).

As files' attributes will be preserved, in most cases you will need to change the files' ownership to mysql before starting the database server, as they will be owned by the user who created the backup:

```
$ chown -R mysql:mysql /var/lib/mysql
```

Also note that all of these operations will be done as the user calling **innobackupex**, you will need write permissions on the server's *datadir*.

Other Types of Backup

Incremental Backups with innobackupex

As not all information changes between each backup, the incremental backup strategy uses this to reduce the storage needs and the duration of making a backup.

This can be done because each *InnoDB* page has a log sequence number, *LSN*, which acts as a version number of the entire database. Every time the database is modified, this number gets incremented.

An incremental backup copies all pages since a specific *LSN*.

Once the pages have been put together in their respective order, applying the logs will recreate the process that affected the database, yielding the data at the moment of the most recently created backup.

Creating an Incremental Backups with `innobackupex`

First, you need to make a full backup as the BASE for subsequent incremental backups:

```
$ innobackupex /data/backups
```

This will create a timestamped directory in `/data/backups`. Assuming that the backup is done last day of the month, `BASEDIR` would be `/data/backups/2013-03-31_23-01-18`, for example.

Note: You can use the *`innobackupex --no-timestamp`* option to override this behavior and the backup will be created in the given directory.

If you check at the `xtrabackup-checkpoints` file in `BASE-DIR`, you should see something like:

```
backup_type = full-backuped
from_lsn = 0
to_lsn = 1626007
last_lsn = 1626007
compact = 0
recover_binlog_info = 1
```

To create an incremental backup the next day, use *`innobackupex --incremental`* and provide the BASEDIR:

```
$ innobackupex --incremental /data/backups --incremental-basedir=BASEDIR
```

Another timestamped directory will be created in `/data/backups`, in this example, `/data/backups/2013-04-01_23-01-18` containing the incremental backup. We will call this `INCREMENTAL-DIR-1`.

If you check at the `xtrabackup-checkpoints` file in `INCREMENTAL-DIR-1`, you should see something like:

```
backup_type = incremental
from_lsn = 1626007
to_lsn = 4124244
last_lsn = 4124244
compact = 0
recover_binlog_info = 1
```

Creating another incremental backup the next day will be analogous, but this time the previous incremental one will be base:

```
$ innobackupex --incremental /data/backups --incremental-basedir=INCREMENTAL-DIR-1
```

Yielding (in this example) `/data/backups/2013-04-02_23-01-18`. We will use `INCREMENTAL-DIR-2` instead for simplicity.

At this point, the `xtrabackup-checkpoints` file in `INCREMENTAL-DIR-2` should contain something like:

```
backup_type = incremental
from_lsn = 4124244
to_lsn = 6938371
last_lsn = 7110572
compact = 0
recover_binlog_info = 1
```

As it was said before, an incremental backup only copy pages with a *LSN* greater than a specific value. Providing the *LSN* would have produced directories with the same data inside:

```
innobackupex --incremental /data/backups --incremental-lsn=4124244
innobackupex --incremental /data/backups --incremental-lsn=6938371
```

This is a very useful way of doing an incremental backup, since not always the base or the last incremental will be available in the system.

Warning: This procedure only affects *XtraDB* or *InnoDB*-based tables. Other tables with a different storage engine, e.g. *MyISAM*, will be copied entirely each time an incremental backup is performed.

Preparing an Incremental Backup with `innobackupex`

Preparing incremental backups is a bit different than full backups. This is, perhaps, the stage where more attention is needed:

- First, **only the committed transactions must be replayed on each backup**. This will merge the base full backup with the incremental ones.
- Then, the uncommitted transaction must be rolled back in order to have a ready-to-use backup.

If you replay the committed transactions **and** rollback the uncommitted ones on the base backup, you will not be able to add the incremental ones. If you do this on an incremental one, you won't be able to add data from that moment and the remaining increments.

Having this in mind, the procedure is very straight-forward using the *`innobackupex --redo-only`* option, starting with the base backup:

```
innobackupex --apply-log --redo-only BASE-DIR
```

You should see an output similar to:

```
160103 22:00:12 InnoDB: Shutdown completed; log sequence number 4124244
160103 22:00:12 innobackupex: completed OK!
```

Then, the first incremental backup can be applied to the base backup, by issuing:

```
innobackupex --apply-log --redo-only BASE-DIR --incremental-dir=INCREMENTAL-DIR-1
```

You should see an output similar to the previous one but with corresponding *LSN*:

```
160103 22:08:43 InnoDB: Shutdown completed; log sequence number 6938371
160103 22:08:43 innobackupex: completed OK!
```

If no *`innobackupex --incremental-dir`* is set, **`innobackupex`** will use the most recent subdirectory created in the basedir.

At this moment, `BASE-DIR` contains the data up to the moment of the first incremental backup. Note that the full data will always be in the directory of the base backup, as we are appending the increments to it.

Repeat the procedure with the second one:

```
$ innobackupex --apply-log BASE-DIR --incremental-dir=INCREMENTAL-DIR-2
```

If the *completed OK!* message was shown, the final data will be in the base backup directory, `BASE-DIR`.

Note: *innobackupex --redo-only* should be used when merging all incrementals except the last one. That's why the previous line doesn't contain the *innobackupex --redo-only* option. Even if the *innobackupex --redo-only* was used on the last step, backup would still be consistent but in that case server would perform the rollback phase.

You can use this procedure to add more increments to the base, as long as you do it in the chronological order that the backups were done. If you merge the incrementals in the wrong order, the backup will be useless. If you have doubts about the order that they must be applied, you can check the file `xtrabackup_checkpoints` at the directory of each one, as shown in the beginning of this section.

Once you merge the base with all the increments, you can prepare it to roll back the uncommitted transactions:

```
$ innobackupex --apply-log BASE-DIR
```

Now your backup is ready to be used immediately after restoring it. This preparation step is optional. However, if you restore without doing the prepare, the database server will begin to rollback uncommitted transactions, the same work it would do if a crash had occurred. This results in delay as the database server starts, and you can avoid the delay if you do the prepare.

Note that the `iblog*` files will not be created by **innobackupex**, if you want them to be created, use **xtrabackup --prepare** on the directory. Otherwise, the files will be created by the server once started.

Restoring Incremental Backups with `innobackupex`

After preparing the incremental backups, the base directory contains the same data as the full backup. For restoring it, you can use the *xtrabackup --copy-back* parameter:

```
$ xtrabackup --copy-back --target-dir=BASE-DIR
```

If the incremental backup was created using the *xtrabackup --compress* option, then you need to run *xtrabackup --decompress* followed by *xtrabackup --copy-back*.

```
$ xtrabackup --decompress --target-dir=BASE-DIR
$ xtrabackup --copy-back --target-dir=BASE-DIR
```

You may have to change the ownership as detailed on *Restoring a Full Backup with innobackupex*.

Incremental Streaming Backups using xbstream and tar

Incremental streaming backups can be performed with the *xbstream* streaming option. Currently backups are packed in custom **xbstream** format. With this feature, you need to take a BASE backup as well.

Taking a base backup

```
$ innobackupex /data/backups
```

Taking a local backup

```
$ innobackupex --incremental --incremental-lsn=LSN-number --stream=xbstream ./ >␣
↪incremental.xbstream
```

Unpacking the backup

```
$ xbstream -x < incremental.xbstream
```

Taking a local backup and streaming it to the remote server and unpacking it

```
$ innobackupex  --incremental --incremental-lsn=LSN-number --stream=xbstream ./ | /
ssh user@hostname " cat - | xbstream -x -C > /backup-dir/"
```

Partial Backups

Percona XtraBackup features partial backups, which means that you may backup only some specific tables or databases. The tables you back up must be in separate tablespaces, as a result of being created or altered after you enabled the *innodb_file_per_table* option on the server.

There is only one caveat about partial backups: do not copy back the prepared backup. Restoring partial backups should be done by importing the tables, not by using the traditional *innobackupex --copy-back* option. Although there are some scenarios where restoring can be done by copying back the files, this may be lead to database inconsistencies in many cases and it is not the recommended way to do it.

Creating Partial Backups

There are three ways of specifying which part of the whole data will be backed up: regular expressions (*innobackupex --include*), enumerating the tables in a file (*innobackupex --tables-file*) or providing a list of databases (*innobackupex --databases*).

Using `innobackupex --include`

The regular expression provided to this will be matched against the fully qualified table name, including the database name, in the form `databasename.tablename`.

For example,

```
$ innobackupex --include='^mydatabase[.]mytable' /path/to/backup
```

The command above will create a timestamped directory with the usual files that **innobackupex** creates, but only the data files related to the tables matched.

Note that this option is passed to *xtrabackup --tables* and is matched against each table of each database, the directories of each database will be created even if they are empty.

Using `innobackupex --tables-file`

The text file provided (the path) to this option can contain multiple table names, one per line, in the `databasename.tablename` format.

For example,

```
$ echo "mydatabase.mytable" > /tmp/tables.txt
$ innobackupex --tables-file=/tmp/tables.txt /path/to/backup
```

The command above will create a timestamped directory with the usual files that **innobackupex** creates, but only containing the data-files related to the tables specified in the file.

This option is passed to *xtrabackup --tables-file* and, unlike the *--tables* option, only directories of databases of the selected tables will be created.

Using innobackupex --databases

This option accepts either a space-separated list of the databases and tables to backup - in the `databasename[.tablename]` form - or a file containing the list at one element per line.

For example,

```
$ innobackupex --databases="mydatabase.mytable mysql" /path/to/backup
```

The command above will create a timestamped directory with the usual files that **innobackupex** creates, but only containing the data-files related to `mytable` in the `mydatabase` directory and the `mysql` directory with the entire `mysql` database.

Preparing Partial Backups

For preparing partial backups, the procedure is analogous to *restoring individual tables* : apply the logs and use the *innobackupex --export* option:

```
$ innobackupex --apply-log --export /path/to/partial/backup
```

You may see warnings in the output about tables that don't exist. This is because *InnoDB* -based engines stores its data dictionary inside the tablespace files besides the *.frm* files. **innobackupex** will use **xtrabackup** to remove the missing tables (those who weren't selected in the partial backup) from the data dictionary in order to avoid future warnings or errors:

```
111225  0:54:06  InnoDB: Error: table 'mydatabase/mytablenotincludedinpartialb'
InnoDB: in InnoDB data dictionary has tablespace id 6,
InnoDB: but tablespace with that id or name does not exist. It will be removed from␣
↪data dictionary.
```

You should also see the notification of the creation of a file needed for importing (*.exp* file) for each table included in the partial backup:

```
xtrabackup: export option is specified.
xtrabackup: export metadata of table 'employees/departments' to file `.//departments.
↪exp` (2 indexes)
xtrabackup:     name=PRIMARY, id.low=80, page=3
xtrabackup:     name=dept_name, id.low=81, page=4
```

Note that you can use the *innobackupex --export* option with *innobackupex --apply-log* to an already-prepared backup in order to create the *.exp* files.

Finally, check for the confirmation message in the output:

```
111225 00:54:18  innobackupex: completed OK!
```

Restoring Partial Backups

Restoring should be done by *restoring individual tables* in the partial backup to the server.

It can also be done by copying back the prepared backup to a "clean" *datadir* (in that case, make sure to include the `mysql` database). System database can be created with:

```
$ sudo mysql_install_db --user=mysql
```

Encrypted Backups

Percona XtraBackup has implemented support for encrypted backups. It can be used to encrypt/decrypt local or streaming backup with *xbstream* option (streaming tar backups are not supported) in order to add another layer of protection to the backups. Encryption is done with the `libgcrypt` library.

Creating Encrypted Backups

To make an encrypted backup following options need to be specified (options *innobackupex --encrypt-key* and *innobackupex --encrypt-key-file* are mutually exclusive, i.e. just one of them needs to be provided):

- *innobackupex --encrypt*
- *innobackupex --encrypt-key*
- *innobackupex --encrypt-key-file*

Both *innobackupex --encrypt-key* option and *innobackupex --encrypt-key-file* option can be used to specify the encryption key. Encryption key can be generated with a command like:

```
$ openssl rand -base64 24
```

Example output of that command should look like this:

```
GCHFLrDFVx6UAsRb88uLVbAVWbK+Yzfs
```

This value then can be used as the encryption key

Using `innobackupex --encrypt-key`

Example of the innobackupex command using the *innobackupex --encrypt-key* should look like this

```
$ innobackupex --encrypt=AES256 --encrypt-key="GCHFLrDFVx6UAsRb88uLVbAVWbK+Yzfs" /
↪data/backups
```

Using `innobackupex --encrypt-key-file`

Example of the innobackupex command using the *innobackupex --encrypt-key-file* should look like this

```
$ innobackupex --encrypt=AES256 --encrypt-key-file=/data/backups/keyfile /data/backups
```

Note: Depending on the text editor used for making the `KEYFILE`, text file in some cases can contain the CRLF and this will cause the key size to grow and thus making it invalid. Suggested way to do this would be to create the file with: `echo -n "GCHFLrDFVx6UAsRb88uLVbAVWbK+Yzfs" > /data/backups/keyfile`

Both of these examples will create a timestamped directory in `/data/backups` containing the encrypted backup.

Note: You can use the *innobackupex --no-timestamp* option to override this behavior and the backup will be created in the given directory.

Optimizing the encryption process

Two new options have been introduced with the encrypted backups that can be used to speed up the encryption process. These are *innobackupex --encrypt-threads* and *innobackupex --encrypt-chunk-size*. By using the *innobackupex --encrypt-threads* option multiple threads can be specified to be used for encryption in parallel. Option *innobackupex --encrypt-chunk-size* can be used to specify the size (in bytes) of the working encryption buffer for each encryption thread (default is 64K).

Decrypting Encrypted Backups

Backups can be decrypted with *The xbcrypt binary*. The following one-liner can be used to encrypt the whole folder:

```
$ for i in `find . -iname "*\.xbcrypt"`; do xbcrypt -d --encrypt-key-file=/root/
↪secret_key --encrypt-algo=AES256 < $i > $(dirname $i)/$(basename $i .xbcrypt) && rm
↪$i; done
```

Percona XtraBackup *innobackupex --decrypt* option has been implemented that can be used to decrypt the backups:

```
$ innobackupex --decrypt=AES256 --encrypt-key="GCHFLrDFVx6UAsRb88uLVbAVWbK+Yzfs" /
↪data/backups/2015-03-18_08-31-35/
```

Percona XtraBackup doesn't automatically remove the encrypted files. In order to clean up the backup directory users should remove the `*.xbcrypt` files.

Note: *innobackupex --parallel* can be used with *innobackupex --decrypt* option to decrypt multiple files simultaneously.

When the files have been decrypted backup can be prepared.

Preparing Encrypted Backups

After the backups have been decrypted, they can be prepared the same way as the standard full backups with the *innobackupex --apply-log* option:

```
$ innobackupex --apply-log /data/backups/2015-03-18_08-31-35/
```

Note: *Percona XtraBackup* doesn't automatically remove the encrypted files. In order to clean up the backup directory users should remove the `*.xbcrypt` files.

Restoring Encrypted Backups

innobackupex has a *innobackupex --copy-back* option, which performs the restoration of a backup to the server's *datadir*

```
$ innobackupex --copy-back /path/to/BACKUP-DIR
```

It will copy all the data-related files back to the server's *datadir*, determined by the server's `my.cnf` configuration file. You should check the last line of the output for a success message:

```
innobackupex: Finished copying back files.
150318 11:08:13  innobackupex: completed OK!
```

Other Reading

- The Libgcrypt Reference Manual

Advanced Features

Streaming and Compressing Backups

Streaming mode, supported by *Percona XtraBackup*, sends backup to `STDOUT` in special `tar` or *xbstream* format instead of copying files to the backup directory.

This allows you to use other programs to filter the output of the backup, providing greater flexibility for storage of the backup. For example, compression is achieved by piping the output to a compression utility. One of the benefits of streaming backups and using Unix pipes is that the backups can be automatically encrypted.

To use the streaming feature, you must use the *innobackupex --stream*, providing the format of the stream (`tar` or `xbstream`) and where to store the temporary files:

```
$ innobackupex --stream=tar /tmp
```

innobackupex uses *xbstream* to stream all of the data files to `STDOUT`, in a special `xbstream` format. See *The xbstream binary* for details. After it finishes streaming all of the data files to `STDOUT`, it stops xtrabackup and streams the saved log file too.

When compression is enabled, **xtrabackup** compresses all output data, except the meta and non-InnoDB files which are not compressed, using the specified compression algorithm. The only currently supported algorithm is `quicklz`. The resulting files have the qpress archive format, i.e. every *.qp file produced by xtrabackup is essentially a one-file qpress archive and can be extracted and uncompressed by the qpress file archiver which is available from *Percona Software repositories*.

Using *xbstream* as a stream option, backups can be copied and compressed in parallel which can significantly speed up the backup process. In case backups were both compressed and encrypted, they'll need to decrypted first in order to be uncompressed.

Examples using xbstream

Store the complete backup directly to a single file:

```
$ innobackupex --stream=xbstream /root/backup/ > /root/backup/backup.xbstream
```

To stream and compress the backup:

```
$ innobackupex --stream=xbstream --compress /root/backup/ > /root/backup/backup.
↪xbstream
```

To unpack the backup to the /root/backup/ directory:

```
$ xbstream -x <  backup.xbstream -C /root/backup/
```

To send the compressed backup to another host and unpack it:

```
$ innobackupex --compress --stream=xbstream /root/backup/ | ssh user@otherhost
↪"xbstream -x -C /root/backup/"
```

Examples using tar

Store the complete backup directly to a tar archive:

```
$ innobackupex --stream=tar /root/backup/ > /root/backup/out.tar
```

To send the tar archive to another host:

```
$ innobackupex --stream=tar ./ | ssh user@destination \ "cat - > /data/backups/backup.
↪tar"
```

Warning: To extract *Percona XtraBackup*'s archive you **must** use **`tar`** with `-i` option:

```
.. code-block:: bash
```

$ tar -xizf backup.tar.gz

Compress with your preferred compression tool:

```
$ innobackupex --stream=tar ./ | gzip - > backup.tar.gz
$ innobackupex --stream=tar ./ | bzip2 - > backup.tar.bz2
```

Note that the streamed backup will need to be prepared before restoration. Streaming mode does not prepare the backup.

Taking Backups in Replication Environments

There are options specific to back up from a replication slave.

`innobackupex --slave-info`

This option is useful when backing up a replication slave server. It prints the binary log position and name of the master server. It also writes this information to the `xtrabackup_slave_info` file as a `CHANGE MASTER` statement.

This is useful for setting up a new slave for this master can be set up by starting a slave server on this backup and issuing the statement saved in the `xtrabackup_slave_info` file. More details of this procedure can be found in *How to setup a slave for replication in 6 simple steps with Percona XtraBackup*.

`innobackupex --safe-slave-backup`

In order to assure a consistent replication state, this option stops the slave SQL thread and wait to start backing up until `Slave_open_temp_tables` in `SHOW STATUS` is zero. If there are no open temporary tables, the backup will take place, otherwise the SQL thread will be started and stopped until there are no open temporary tables. The backup will fail if `Slave_open_temp_tables` does not become zero after *`innobackupex --safe-slave-backup-timeout`* seconds (defaults to 300 seconds). The slave SQL thread will be restarted when the backup finishes.

Using this option is always recommended when taking backups from a slave server.

Warning: Make sure your slave is a true replica of the master before using it as a source for backup. A good tool to validate a slave is pt-table-checksum.

Accelerating the backup process

Accelerating with `innobackupex --parallel` copy and *–compress-threads*

When performing a local backup or the streaming backup with *xbstream* option, multiple files can be copied concurrently by using the *`innobackupex --parallel`* option. This option specifies the number of threads created by **`xtrabackup`** to copy data files.

To take advantage of this option either the multiple tablespaces option must be enabled (*innodb_file_per_table*) or the shared tablespace must be stored in multiple *ibdata* files with the *innodb_data_file_path* option. Having multiple files for the database (or splitting one into many) doesn't have a measurable impact on performance.

As this feature is implemented **at a file level**, concurrent file transfer can sometimes increase I/O throughput when doing a backup on highly fragmented data files, due to the overlap of a greater number of random read requests. You should consider tuning the filesystem also to obtain the maximum performance (e.g. checking fragmentation).

If the data is stored on a single file, this option will have no effect.

To use this feature, simply add the option to a local backup, for example:

```
$ innobackupex --parallel=4 /path/to/backup
```

By using the *xbstream* in streaming backups you can additionally speed up the compression process by using the *`innobackupex --compress-threads`* option. This option specifies the number of threads created by **`xtrabackup`** for for parallel data compression. The default value for this option is 1.

To use this feature, simply add the option to a local backup, for example

```
$ innobackupex --stream=xbstream --compress --compress-threads=4 ./ > backup.xbstream
```

Before applying logs, compressed files will need to be uncompressed.

Accelerating with `innobackupex --rsync`

In order to speed up the backup process and to minimize the time `FLUSH TABLES WITH READ LOCK` is blocking the writes, option *`innobackupex --rsync`* should be used. When this option is specified, **innobackupex** uses `rsync` to copy all non-InnoDB files instead of spawning a separate `cp` for each file, which can be much faster for servers with a large number of databases or tables. **innobackupex** will call the `rsync` twice, once before the `FLUSH TABLES WITH READ LOCK` and once during to minimize the time the read lock is being held. During the second `rsync` call, it will only synchronize the changes to non-transactional data (if any) since the first call performed before the `FLUSH TABLES WITH READ LOCK`. Note that *Percona XtraBackup* will use Backup locks where available as a lightweight alternative to `FLUSH TABLES WITH READ LOCK`. This feature is available in *Percona Server for MySQL* 5.6+. *Percona XtraBackup* uses this automatically to copy non-InnoDB data to avoid blocking DML queries that modify InnoDB tables.

Note: This option cannot be used together with the *`innobackupex --stream`* option.

Throttling backups with `innobackupex`

Although **innobackupex** does not block your database's operation, any backup can add load to the system being backed up. On systems that do not have much spare I/O capacity, it might be helpful to throttle the rate at which **innobackupex** reads and writes *InnoDB* data. You can do this with the *`innobackupex --throttle`* option.

This option is passed directly to **xtrabackup** binary and only limits the operations on the logs and files of *InnoDB* tables. It doesn't have an effect on reading or writing files from tables with other storage engine.

One way of checking the current I/O operations at a system is with **iostat** command. See *Throttling Backups* for details of how throttling works.

Note: *`innobackupex --throttle`* option works only during the backup phase, i.e. it will not work with *`innobackupex --apply-log`* and *`innobackupex --copy-back`* options.

The *`innobackupex --throttle`* option is similar to the `--sleep` option in `mysqlbackup` and should be used instead of it, as `--sleep` will be ignored.

Restoring Individual Tables

In server versions prior to 5.6, it is not possible to copy tables between servers by copying the files, even with *innodb_file_per_table*. However, with the *Percona XtraBackup*, you can export individual tables from any *InnoDB* database, and import them into *Percona Server for MySQL* with *XtraDB* or *MySQL* 5.6 (The source doesn't have to be *XtraDB* or or *MySQL* 5.6, but the destination does). This only works on individual *.ibd* files, and cannot export a table that is not contained in its own *.ibd* file.

Note: If you're running *Percona Server for MySQL* version older than 5.5.10-20.1, variable innodb_expand_import should be used instead of innodb_import_table_from_xtrabackup.

Exporting tables

Exporting is done in the preparation stage, not at the moment of creating the backup. Once a full backup is created, prepare it with the *`innobackupex --export`* option:

```
$ innobackupex --apply-log --export /path/to/backup
```

This will create for each *InnoDB* with its own tablespace a file with *.exp* extension. An output of this procedure would contain:

```
..
xtrabackup: export option is specified.
xtrabackup: export metadata of table 'mydatabase/mytable' to file
`./mydatabase/mytable.exp` (1 indexes)
..
```

Now you should see a *.exp* file in the target directory:

```
$ find /data/backups/mysql/ -name export_test.*
/data/backups/mysql/test/export_test.exp
/data/backups/mysql/test/export_test.ibd
/data/backups/mysql/test/export_test.cfg
```

These three files are all you need to import the table into a server running *Percona Server for MySQL* with *XtraDB* or *MySQL* 5.6.

Note: *MySQL* uses `.cfg` file which contains *InnoDB* dictionary dump in special format. This format is different from the `.exp` one which is used in *XtraDB* for the same purpose. Strictly speaking, a `.cfg` file is **not** required to import a tablespace to *MySQL* 5.6 or *Percona Server for MySQL* 5.6. A tablespace will be imported successfully even if it is from another server, but *InnoDB* will do schema validation if the corresponding `.cfg` file is present in the same directory.

Each *.exp* (or `.cfg`) file will be used for importing that table.

Note: InnoDB does a slow shutdown (i.e. full purge + change buffer merge) on –export, otherwise the tablespaces wouldn't be consistent and thus couldn't be imported. All the usual performance considerations apply: sufficient buffer pool (i.e. –use-memory, 100MB by default) and fast enough storage, otherwise it can take a prohibitive amount of time for export to complete.

Importing tables

To import a table to other server, first create a new table with the same structure as the one that will be imported at that server:

```
OTHERSERVER|mysql> CREATE TABLE mytable (...) ENGINE=InnoDB;
```

then discard its tablespace:

```
OTHERSERVER|mysql> ALTER TABLE mydatabase.mytable DISCARD TABLESPACE;
```

Next, copy `mytable.ibd` and `mytable.exp` (or `mytable.cfg` if importing to *MySQL* 5.6) files to database's home, and import its tablespace:

```
OTHERSERVER|mysql> ALTER TABLE mydatabase.mytable IMPORT TABLESPACE;
```

Set the owner and group of the files:

```
$ chown -R mysql:mysql /datadir/db_name/table_name.*
```

After running this command, data in the imported table will be available.

Point-In-Time recovery

Recovering up to particular moment in database's history can be done with **innobackupex** and the binary logs of the server.

Note that the binary log contains the operations that modified the database from a point in the past. You need a full *datadir* as a base, and then you can apply a series of operations from the binary log to make the data match what it was at the point in time you want.

For taking the snapshot, we will use **innobackupex** for a full backup:

```
$ innobackupex /path/to/backup --no-timestamp
```

(the *innobackupex --no-timestamp* option is for convenience in this example) and we will prepare it to be ready for restoration:

```
$ innobackupex --apply-log /path/to/backup
```

For more details on these procedures, see *Creating a Backup with innobackupex* and *Preparing a Full Backup with innobackupex*.

Now, suppose that time has passed, and you want to restore the database to a certain point in the past, having in mind that there is the constraint of the point where the snapshot was taken.

To find out what is the situation of binary logging in the server, execute the following queries:

```
mysql> SHOW BINARY LOGS;
+------------------+-----------+
| Log_name         | File_size |
+------------------+-----------+
| mysql-bin.000001 |       126 |
| mysql-bin.000002 |      1306 |
| mysql-bin.000003 |       126 |
| mysql-bin.000004 |       497 |
+------------------+-----------+
```

and

```
mysql> SHOW MASTER STATUS;
+------------------+----------+--------------+------------------+
| File             | Position | Binlog_Do_DB | Binlog_Ignore_DB |
+------------------+----------+--------------+------------------+
| mysql-bin.000004 |      497 |              |                  |
+------------------+----------+--------------+------------------+
```

The first query will tell you which files contain the binary log and the second one which file is currently being used to record changes, and the current position within it. Those files are stored usually in the *datadir* (unless other location is specified when the server is started with the `--log-bin=` option).

To find out the position of the snapshot taken, see the `xtrabackup_binlog_info` at the backup's directory:

```
$ cat /path/to/backup/xtrabackup_binlog_info
mysql-bin.000003      57
```

This will tell you which file was used at moment of the backup for the binary log and its position. That position will be the effective one when you restore the backup:

```
$ innobackupex --copy-back /path/to/backup
```

As the restoration will not affect the binary log files (you may need to adjust file permissions, see *Restoring a Full Backup with innobackupex*), the next step is extracting the queries from the binary log with **mysqlbinlog** starting from the position of the snapshot and redirecting it to a file

```
$ mysqlbinlog /path/to/datadir/mysql-bin.000003 /path/to/datadir/mysql-bin.000004 \
    --start-position=57 > mybinlog.sql
```

Note that if you have multiple files for the binary log, as in the example, you have to extract the queries with one process, as shown above.

Inspect the file with the queries to determine which position or date corresponds to the point-in-time wanted. Once determined, pipe it to the server. Assuming the point is `11-12-25 01:00:00`:

```
$ mysqlbinlog /path/to/datadir/mysql-bin.000003 /path/to/datadir/mysql-bin.000004 \
    --start-position=57 --stop-datetime="11-12-25 01:00:00" | mysql -u root -p
```

and the database will be rolled forward up to that Point-In-Time.

Improved `FLUSH TABLES WITH READ LOCK` handling

When taking backups, `FLUSH TABLES WITH READ LOCK` is being used before the non-InnoDB files are being backed up to ensure backup is being consistent. `FLUSH TABLES WITH READ LOCK` can be run even though there may be a running query that has been executing for hours. In this case everything will be locked up in `Waiting for table flush` or `Waiting for master to send event` states. Killing the `FLUSH TABLES WITH READ LOCK` does not correct this issue either. In this case the only way to get the server operating normally again is to kill off the long running queries that blocked it to begin with. This means that if there are long running queries `FLUSH TABLES WITH READ LOCK` can get stuck, leaving server in read-only mode until waiting for these queries to complete.

Note: All described in this section has no effect when backup locks are used. *Percona XtraBackup* will use Backup locks where available as a lightweight alternative to `FLUSH TABLES WITH READ LOCK`. This feature is available in *Percona Server for MySQL* 5.6+. *Percona XtraBackup* uses this automatically to copy non-InnoDB data to avoid blocking DML queries that modify InnoDB tables.

In order to prevent this from happening two things have been implemented:

- **innobackupex** can wait for a good moment to issue the global lock.
- **innobackupex** can kill all or only SELECT queries which are preventing the global lock from being acquired

Waiting for queries to finish

Good moment to issue a global lock is the moment when there are no long queries running. But waiting for a good moment to issue the global lock for extended period of time isn't always good approach, as it can extend the time needed for backup to take place. To prevent **innobackupex** from waiting to issue `FLUSH TABLES WITH READ LOCK` for too long, new option has been implemented: *`innobackupex --ftwrl-wait-timeout`* option can be used to limit the waiting time. If the good moment to issue the lock did not happen during this time, **innobackupex** will give up and exit with an error message and backup will not be taken. Zero value for this option turns off the feature (which is default).

Another possibility is to specify the type of query to wait on. In this case *innobackupex --ftwrl-wait-query-type*. Possible values are `all` and `update`. When `all` is used **innobackupex** will wait for all long running queries (execution time longer than allowed by *innobackupex --ftwrl-wait-threshold*) to finish before running the `FLUSH TABLES WITH READ LOCK`. When `update` is used **innobackupex** will wait on `UPDATE/ALTER/REPLACE/INSERT` queries to finish.

Although time needed for specific query to complete is hard to predict, we can assume that queries that are running for a long time already will likely not be completed soon, and queries which are running for a short time will likely be completed shortly. **innobackupex** can use the value of *innobackupex --ftwrl-wait-threshold* option to specify which query is long running and will likely block global lock for a while. In order to use this option xtrabackup user should have `PROCESS` and `SUPER` privileges.

Killing the blocking queries

Second option is to kill all the queries which prevent global lock from being acquired. In this case all the queries which run longer than `FLUSH TABLES WITH READ LOCK` are possible blockers. Although all queries can be killed, additional time can be specified for the short running queries to complete. This can be specified by *innobackupex --kill-long-queries-timeout* option. This option specifies the time for queries to complete, after the value is reached, all the running queries will be killed. Default value is zero, which turns this feature off.

innobackupex --kill-long-query-type option can be used to specify all or only `SELECT` queries that are preventing global lock from being acquired. In order to use this option xtrabackup user should have `PROCESS` and `SUPER` privileges.

Options summary

- *innobackupex --ftwrl-wait-timeout* (seconds) - how long to wait for a good moment. Default is 0, not to wait.
- *innobackupex --ftwrl-wait-query-type* - which long queries should be finished before `FLUSH TABLES WITH READ LOCK` is run. Default is all.
- *innobackupex --ftwrl-wait-threshold* (seconds) - how long query should be running before we consider it long running and potential blocker of global lock.
- *innobackupex --kill-long-queries-timeout* (seconds) - how many time we give for queries to complete after `FLUSH TABLES WITH READ LOCK` is issued before start to kill. Default if `0`, not to kill.
- *innobackupex --kill-long-query-type* - which queries should be killed once `kill-long-queries-timeout` has expired.

Example

Running the **innobackupex** with the following options will cause **innobackupex** to spend no longer than 3 minutes waiting for all queries older than 40 seconds to complete.

```
$ innobackupex --ftwrl-wait-threshold=40 --ftwrl-wait-query-type=all --ftwrl-wait-
↪timeout=180 --kill-long-queries-timeout=20 --kill-long-query-type=all /data/backups/
```

After `FLUSH TABLES WITH READ LOCK` is issued, **innobackupex** will wait 20 seconds for lock to be acquired. If lock is still not acquired after 20 seconds, it will kill all queries which are running longer that the `FLUSH TABLES WITH READ LOCK`.

Store backup history on the server

Percona XtraBackup supports storing the backups history on the server. This feature was implemented in *Percona XtraBackup* 2.2. Storing backup history on the server was implemented to provide users with additional information about backups that are being taken. Backup history information will be stored in the *PERCONA_SCHEMA.XTRABACKUP_HISTORY* table.

To use this feature three new **innobackupex** options have been implemented:

- *innobackupex --history* =<name> : This option enables the history feature and allows the user to specify a backup series name that will be placed within the history record.
- *innobackupex --incremental-history-name* =<name> : This option allows an incremental backup to be made based on a specific history series by name. **innobackupex** will search the history table looking for the most recent (highest `to_lsn`) backup in the series and take the `to_lsn` value to use as it's starting lsn. This is mutually exclusive with *innobackupex --incremental-history-uuid*, *innobackupex --incremental-basedir* and *innobackupex --incremental-lsn* options. If no valid LSN can be found (no series by that name) **innobackupex** will return with an error.
- *innobackupex --incremental-history-uuid* =<uuid> : Allows an incremental backup to be made based on a specific history record identified by UUID. **innobackupex** will search the history table looking for the record matching UUID and take the `to_lsn` value to use as it's starting LSN. This options is mutually exclusive with *innobackupex --incremental-basedir*, *innobackupex --incremental-lsn* and *innobackupex --incremental-history-name* options. If no valid LSN can be found (no record by that UUID or missing `to_lsn`), **innobackupex** will return with an error.

Note: Backup that's currently being performed will **NOT** exist in the xtrabackup_history table within the resulting backup set as the record will not be added to that table until after the backup has been taken.

If you want access to backup history outside of your backup set in the case of some catastrophic event, you will need to either perform a `mysqldump`, partial backup or `SELECT *` on the history table after **innobackupex** completes and store the results with you backup set.

Privileges

User performing the backup will need following privileges:

- `CREATE` privilege in order to create the *PERCONA_SCHEMA.xtrabackup_history* database and table.
- `INSERT` privilege in order to add history records to the *PERCONA_SCHEMA.xtrabackup_history* table.
- `SELECT` privilege in order to use *innobackupex --incremental-history-name* or *innobackupex --incremental-history-uuid* in order for the feature to look up the `innodb_to_lsn` values in the *PERCONA_SCHEMA.xtrabackup_history* table.

PERCONA_SCHEMA.XTRABACKUP_HISTORY table

This table contains the information about the previous server backups. Information about the backups will only be written if the backup was taken with *innobackupex --history* option.

Column Name	Description
uuid	Unique backup id
name	User provided name of backup series. There may be multiple entries with the same name used to identify related backups in a series.
tool_name	Name of tool used to take backup
tool_command	Exact command line given to the tool with –password and –encryption_key obfuscated
tool_version	Version of tool used to take backup
ib-backup_version	Version of the xtrabackup binary used to take backup
server_version	Server version on which backup was taken
start_time	Time at the start of the backup
end_time	Time at the end of the backup
lock_time	Amount of time, in seconds, spent calling and holding locks for `FLUSH TABLES WITH READ LOCK`
binlog_pos	Binlog file and position at end of `FLUSH TABLES WITH READ LOCK`
inn-odb_from_lsn	LSN at beginning of backup which can be used to determine prior backups
inn-odb_to_lsn	LSN at end of backup which can be used as the starting lsn for the next incremental
partial	Is this a partial backup, if `N` that means that it's the full backup
incremental	Is this an incremental backup
format	Description of result format (`file`, `tar`, `xbstream`)
compressed	Is this a compressed backup
encrypted	Is this an encrypted backup

Limitations

- *`innobackupex --history`* option must be specified only on the innobackupex command line and not within a configuration file in order to be effective.
- *`innobackupex --incremental-history-name`* and *`innobackupex --incremental-history-uuid`* options must be specified only on the innobackupex command line and not within a configuration file in order to be effective.

Implementation

How `innobackupex` Works

From *Percona XtraBackup* version 2.3 **`innobackupex`** is has been rewritten in *C* and set up as a symlink to the **`xtrabackup`**. **`innobackupex`** supports all features and syntax as 2.2 version did, but it is now deprecated and will be removed in next major release. Syntax for new features will not be added to the innobackupex, only to the xtrabackup.

The following describes the rationale behind **`innobackupex`** actions.

Making a Backup

If no mode is specified, **`innobackupex`** will assume the backup mode.

By default, it runs **`xtrabackup`** and lets it copy the InnoDB data files. When **`xtrabackup`** finishes that, **`innobackupex`** sees it create the `xtrabackup_suspended_2` file and executes `FLUSH TABLES WITH`

`READ LOCK`. **xtrabackup** will use Backup locks where available as a lightweight alternative to `FLUSH TABLES WITH READ LOCK`. This feature is available in *Percona Server for MySQL* 5.6+. *Percona XtraBackup* uses this automatically to copy non-InnoDB data to avoid blocking DML queries that modify InnoDB tables. Then it begins copying the rest of the files.

innobackupex will then check *MySQL* variables to determine which features are supported by server. Special interest are backup locks, changed page bitmaps, GTID mode, etc. If everything goes well, the binary is started as a child process.

innobackupex will wait for slaves in a replication setup if the option *innobackupex --safe-slave-backup* is set and will flush all tables with **READ LOCK**, preventing all *MyISAM* tables from writing (unless option *innobackupex --no-lock* is specified).

Note: Locking is done only for MyISAM and other non-InnoDB tables, and only **after** *Percona XtraBackup* is finished backing up all InnoDB/XtraDB data and logs. *Percona XtraBackup* will use `backup locks` where available as a lightweight alternative to `FLUSH TABLES WITH READ LOCK`. This feature is available in *Percona Server for MySQL* 5.6+. *Percona XtraBackup* uses this automatically to copy non-InnoDB data to avoid blocking DML queries that modify InnoDB tables.

Once this is done, the backup of the files will begin. It will backup *.frm*, *.MRG*, *.MYD*, *.MYI*, *.TRG*, *.TRN*, *.ARM*, *.ARZ*, *.CSM*, *.CSV*, `.par`, and *.opt* files.

When all the files are backed up, it resumes **ibbackup** and wait until it finishes copying the transactions done while the backup was done. Then, the tables are unlocked, the slave is started (if the option *innobackupex --safe-slave-backup* was used) and the connection with the server is closed. Then, it removes the `xtrabackup_suspended_2` file and permits **xtrabackup** to exit.

It will also create the following files in the directory of the backup:

xtrabackup_checkpoints containing the *LSN* and the type of backup;

xtrabackup_binlog_info containing the position of the binary log at the moment of backing up;

xtrabackup_binlog_pos_innodb containing the position of the binary log at the moment of backing up relative to *InnoDB* transactions;

xtrabackup_slave_info containing the MySQL binlog position of the master server in a replication setup via `SHOW SLAVE STATUS` if the *innobackupex --slave-info* option is passed;

backup-my.cnf containing only the `my.cnf` options required for the backup. For example, innodb_data_file_path, innodb_log_files_in_group, innodb_log_file_size, innodb_fast_checksum, innodb_page_size, innodb_log_block_size;

xtrabackup_binary containing the binary used for the backup;

mysql-stderr containing the `STDERR` of **mysqld** during the process and

mysql-stdout containing the `STDOUT` of the server.

Finally, the binary log position will be printed to `STDERR` and **innobackupex** will exit returning 0 if all went OK.

Note that the `STDERR` of **innobackupex** is not written in any file. You will have to redirect it to a file, e.g., `innobackupex OPTIONS 2> backupout.log`.

Restoring a backup

To restore a backup with **innobackupex** the *innobackupex --copy-back* option must be used.

innobackupex will read from the `my.cnf` the variables *datadir*, *innodb_data_home_dir*, *innodb_data_file_path*, *innodb_log_group_home_dir* and check that the directories exist.

It will copy the *MyISAM* tables, indexes, etc. (*.frm*, *.MRG*, *.MYD*, *.MYI*, *.TRG*, *.TRN*, *.ARM*, *.ARZ*, *.CSM*, *.CSV*, `par` and *.opt* files) first, *InnoDB* tables and indexes next and the log files at last. It will preserve file's attributes when copying them, you may have to change the files' ownership to `mysql` before starting the database server, as they will be owned by the user who created the backup.

Alternatively, the *innobackupex --move-back* option may be used to restore a backup. This option is similar to *innobackupex --copy-back* with the only difference that instead of copying files it moves them to their target locations. As this option removes backup files, it must be used with caution. It is useful in cases when there is not enough free disk space to hold both data files and their backup copies.

References

The innobackupex Option Reference

This page documents all of the command-line options for the **innobackupex**.

Options

--apply-log
: Prepare a backup in `BACKUP-DIR` by applying the transaction log file named `xtrabackup_logfile` located in the same directory. Also, create new transaction logs. The InnoDB configuration is read from the file `backup-my.cnf` created by **innobackupex** when the backup was made. innobackupex –apply-log uses InnoDB configuration from `backup-my.cnf` by default, or from –defaults-file, if specified. InnoDB configuration in this context means server variables that affect data format, i.e. `innodb_page_size`, `innodb_log_block_size`, etc. Location-related variables, like `innodb_log_group_home_dir` or *innodb_data_file_path*' are always ignored by –apply-log, so preparing a backup always works with data files from the backup directory, rather than any external ones.

--backup-locks
: This option controls if backup locks should be used instead of `FLUSH TABLES WITH READ LOCK` on the backup stage. The option has no effect when backup locks are not supported by the server. This option is enabled by default, disable with *--no-backup-locks*.

--no-backup-locks
: Explicity disables the option *--backup-locks* which is enabled by default.

--close-files
: Do not keep files opened. This option is passed directly to xtrabackup. When xtrabackup opens tablespace it normally doesn't close its file handle in order to handle the DDL operations correctly. However, if the number of tablespaces is really huge and can not fit into any limit, there is an option to close file handles once they are no longer accessed. *Percona XtraBackup* can produce inconsistent backups with this option enabled. Use at your own risk.

--compress
: This option instructs xtrabackup to compress backup copies of InnoDB data files. It is passed directly to the xtrabackup child process. See the **xtrabackup** *documentation* for details.

--compress-threads=#
: This option specifies the number of worker threads that will be used for parallel compression. It is passed directly to the xtrabackup child process. See the **xtrabackup** *documentation* for details.

--compress-chunk-size=#
: This option specifies the size of the internal working buffer for each compression thread, measured in bytes. It is

passed directly to the xtrabackup child process. The default value is 64K. See the **xtrabackup** *documentation* for details.

--copy-back
Copy all the files in a previously made backup from the backup directory to their original locations. *Percona XtraBackup* *innobackupex --copy-back* option will not copy over existing files unless *innobackupex --force-non-empty-directories* option is specified.

--databases=LIST
This option specifies the list of databases that **innobackupex** should back up. The option accepts a string argument or path to file that contains the list of databases to back up. The list is of the form "database-name1[.table_name1] databasename2[.table_name2] . . .". If this option is not specified, all databases containing *MyISAM* and *InnoDB* tables will be backed up. Please make sure that –databases contains all of the *InnoDB* databases and tables, so that all of the innodb.frm files are also backed up. In case the list is very long, this can be specified in a file, and the full path of the file can be specified instead of the list. (See option –tables-file.)

--decompress
Decompresses all files with the .qp extension in a backup previously made with the *innobackupex --compress* option. The *innobackupex --parallel* option will allow multiple files to be decrypted and/or decompressed simultaneously. In order to decompress, the qpress utility MUST be installed and accessible within the path. *Percona XtraBackup* doesn't automatically remove the compressed files. In order to clean up the backup directory users should remove the `*.qp` files manually.

--decrypt=ENCRYPTION-ALGORITHM
Decrypts all files with the .xbcrypt extension in a backup previously made with –encrypt option. The *innobackupex --parallel* option will allow multiple files to be decrypted and/or decompressed simultaneously.

--defaults-file=[MY.CNF]
This option accepts a string argument that specifies what file to read the default MySQL options from. Must be given as the first option on the command-line.

--defaults-extra-file=[MY.CNF]
This option specifies what extra file to read the default *MySQL* options from before the standard defaults-file. Must be given as the first option on the command-line.

--defaults-group=GROUP-NAME
This option accepts a string argument that specifies the group which should be read from the configuration file. This is needed if you use mysqld_multi. This can also be used to indicate groups other than mysqld and xtrabackup.

--encrypt=ENCRYPTION_ALGORITHM
This option instructs xtrabackup to encrypt backup copies of InnoDB data files using the algorithm specified in the ENCRYPTION_ALGORITHM. It is passed directly to the xtrabackup child process. See the **xtrabackup** *documentation* for more details.

Currently, the following algorithms are supported: `AES128`, `AES192` and `AES256`.

--encrypt-key=ENCRYPTION_KEY
This option instructs xtrabackup to use the given proper length encryption key as the ENCRYPTION_KEY when using the –encrypt option. It is passed directly to the xtrabackup child process. See the **xtrabackup** *documentation* for more details.

It is not recommended to use this option where there is uncontrolled access to the machine as the command line and thus the key can be viewed as part of the process info.

--encrypt-key-file=ENCRYPTION_KEY_FILE
This option instructs xtrabackup to use the encryption key stored in the given ENCRYPTION_KEY_FILE when

using the –encrypt option. It is passed directly to the xtrabackup child process. See the **xtrabackup** *documentation* for more details.

The file must be a simple binary (or text) file that contains exactly the key to be used.

--encrypt-threads=#
This option specifies the number of worker threads that will be used for parallel encryption. It is passed directly to the xtrabackup child process. See the **xtrabackup** *documentation* for more details.

--encrypt-chunk-size=#
This option specifies the size of the internal working buffer for each encryption thread, measured in bytes. It is passed directly to the xtrabackup child process. See the **xtrabackup** *documentation* for more details.

--export
This option is passed directly to *xtrabackup --export* option. It enables exporting individual tables for import into another server. See the **xtrabackup** documentation for details.

--extra-lsndir=DIRECTORY
This option accepts a string argument that specifies the directory in which to save an extra copy of the xtrabackup_checkpoints file. It is passed directly to **xtrabackup**'s *innobackupex --extra-lsndir* option. See the **xtrabackup** documentation for details.

--force-non-empty-directories
When specified, it makes *innobackupex --copy-back* option or *innobackupex --move-back* option transfer files to non-empty directories. No existing files will be overwritten. If –copy-back or –move-back has to copy a file from the backup directory which already exists in the destination directory, it will still fail with an error.

--galera-info
This options creates the xtrabackup_galera_info file which contains the local node state at the time of the backup. Option should be used when performing the backup of Percona-XtraDB-Cluster. Has no effect when backup locks are used to create the backup.

--help
This option displays a help screen and exits.

--history=NAME
This option enables the tracking of backup history in the PERCONA_SCHEMA.xtrabackup_history table. An optional history series name may be specified that will be placed with the history record for the current backup being taken.

--host=HOST
This option accepts a string argument that specifies the host to use when connecting to the database server with TCP/IP. It is passed to the mysql child process without alteration. See **mysql --help** for details.

--ibbackup=IBBACKUP-BINARY
This option specifies which **xtrabackup** binary should be used. The option accepts a string argument. IBBACKUP-BINARY should be the command used to run *Percona XtraBackup*. The option can be useful if the **xtrabackup** binary is not in your search path or working directory. If this option is not specified, **innobackupex** attempts to determine the binary to use automatically.

--include=REGEXP
This option is a regular expression to be matched against table names in databasename.tablename format. It is passed directly to xtrabackup's *xtrabackup --tables* option. See the **xtrabackup** documentation for details.

--incremental
This option tells **xtrabackup** to create an incremental backup, rather than a full one. It is passed to the **xtrabackup** child process. When this option is specified, either *innobackupex --incremental-lsn* or *innobackupex --incremental-basedir* can also be given. If neither option is given, option

innobackupex --incremental-basedir is passed to **xtrabackup** by default, set to the first timestamped backup directory in the backup base directory.

--incremental-basedir=DIRECTORY
This option accepts a string argument that specifies the directory containing the full backup that is the base dataset for the incremental backup. It is used with the *innobackupex --incremental* option.

--incremental-dir=DIRECTORY
This option accepts a string argument that specifies the directory where the incremental backup will be combined with the full backup to make a new full backup. It is used with the *innobackupex --incremental* option.

--incremental-history-name=NAME
This option specifies the name of the backup series stored in the *PERCONA_SCHEMA.xtrabackup_history* history record to base an incremental backup on. Percona Xtrabackup will search the history table looking for the most recent (highest innodb_to_lsn), successful backup in the series and take the to_lsn value to use as the starting lsn for the incremental backup. This will be mutually exclusive with *innobackupex --incremental-history-uuid*, *innobackupex --incremental-basedir* and *innobackupex --incremental-lsn*. If no valid lsn can be found (no series by that name, no successful backups by that name) xtrabackup will return with an error. It is used with the *innobackupex --incremental* option.

--incremental-history-uuid=UUID
This option specifies the UUID of the specific history record stored in the *PERCONA_SCHEMA.xtrabackup_history* to base an incremental backup on. *innobackupex --incremental-history-name*,:optionL‘innobackupex –incremental-basedir‘ and *innobackupex --incremental-lsn*. If no valid lsn can be found (no success record with that uuid) xtrabackup will return with an error. It is used with the *innobackupex --incremental* option.

--incremental-lsn=LSN
This option accepts a string argument that specifies the log sequence number (*LSN*) to use for the incremental backup. It is used with the *innobackupex --incremental* option. It is used instead of specifying *innobackupex --incremental-basedir*. For databases created by *MySQL* and *Percona Server* 5.0-series versions, specify the as two 32-bit integers in high:low format. For databases created in 5.1 and later, specify the LSN as a single 64-bit integer.

--kill-long-queries-timeout=SECONDS
This option specifies the number of seconds innobackupex waits between starting `FLUSH TABLES WITH READ LOCK` and killing those queries that block it. Default is 0 seconds, which means innobackupex will not attempt to kill any queries. In order to use this option xtrabackup user should have `PROCESS` and `SUPER` privileges. Where supported (Percona Server 5.6+) xtrabackup will automatically use Backup Locks as a lightweight alternative to `FLUSH TABLES WITH READ LOCK` to copy non-InnoDB data to avoid blocking DML queries that modify InnoDB tables.

--kill-long-query-type=all|select
This option specifies which types of queries should be killed to unblock the global lock. Default is “all”.

--ftwrl-wait-timeout=SECONDS
This option specifies time in seconds that innobackupex should wait for queries that would block `FLUSH TABLES WITH READ LOCK` before running it. If there are still such queries when the timeout expires, innobackupex terminates with an error. Default is 0, in which case innobackupex does not wait for queries to complete and starts `FLUSH TABLES WITH READ LOCK` immediately. Where supported (Percona Server 5.6+) xtrabackup will automatically use Backup Locks as a lightweight alternative to `FLUSH TABLES WITH READ LOCK` to copy non-InnoDB data to avoid blocking DML queries that modify InnoDB tables.

--ftwrl-wait-threshold=SECONDS
This option specifies the query run time threshold which is used by innobackupex to detect long-running queries with a non-zero value of *innobackupex --ftwrl-wait-timeout*. `FLUSH TABLES WITH READ`

LOCK is not started until such long-running queries exist. This option has no effect if –ftwrl-wait-timeout is 0. Default value is 60 seconds. Where supported (Percona Server 5.6+) xtrabackup will automatically use Backup Locks as a lightweight alternative to FLUSH TABLES WITH READ LOCK to copy non-InnoDB data to avoid blocking DML queries that modify InnoDB tables.

--ftwrl-wait-query-type=all|update
This option specifies which types of queries are allowed to complete before innobackupex will issue the global lock. Default is all.

--log-copy-interval=#
This option specifies time interval between checks done by log copying thread in milliseconds.

--move-back
Move all the files in a previously made backup from the backup directory to their original locations. As this option removes backup files, it must be used with caution.

--no-lock
Use this option to disable table lock with FLUSH TABLES WITH READ LOCK. Use it only if ALL your tables are InnoDB and you **DO NOT CARE** about the binary log position of the backup. This option shouldn't be used if there are any DDL statements being executed or if any updates are happening on non-InnoDB tables (this includes the system MyISAM tables in the *mysql* database), otherwise it could lead to an inconsistent backup. Where supported (Percona Server 5.6+) xtrabackup will automatically use Backup Locks as a lightweight alternative to FLUSH TABLES WITH READ LOCK to copy non-InnoDB data to avoid blocking DML queries that modify InnoDB tables. If you are considering to use *innobackupex --no-lock* because your backups are failing to acquire the lock, this could be because of incoming replication events preventing the lock from succeeding. Please try using *innobackupex --safe-slave-backup* to momentarily stop the replication slave thread, this may help the backup to succeed and you then don't need to resort to using this option. xtrabackup_binlog_info is not created when –no-lock option is used (because SHOW MASTER STATUS may be inconsistent), but under certain conditions xtrabackup_binlog_pos_innodb can be used instead to get consistent binlog coordinates as described in *Working with Binary Logs*.

--no-timestamp
This option prevents creation of a time-stamped subdirectory of the BACKUP-ROOT-DIR given on the command line. When it is specified, the backup is done in BACKUP-ROOT-DIR instead.

--no-version-check
This option disables the version check. If you do not pass this option, the automatic version check is enabled implicitly when **innobackupex** runs in the --backup mode. To disable the version check, you should pass explicitly the --no-version-check option when envoking **innobackupex**.

When the automatic version check is enabled, **innobackupex** performs a version check against the server on the backup stage after creating a server connection. **innobackupex** sends the following information to the server:

- MySQL flavour and version
- Operating system name
- Percona Toolkit version
- Perl version

Each piece of information has a unique identifier. This is a MD5 hash value that Percona Toolkit uses to obtain statistics about how it is used. This is a random UUID; no client information is either collected or stored.

--parallel=NUMBER-OF-THREADS
This option accepts an integer argument that specifies the number of threads the **xtrabackup** child process should use to back up files concurrently. Note that this option works on file level, that is, if you have several .ibd files, they will be copied in parallel. If your tables are stored together in a single tablespace file, it will have no effect. This option will allow multiple files to be decrypted and/or decompressed simultaneously. In

order to decompress, the qpress utility MUST be installed and accessable within the path. This process will remove the original compressed/encrypted files and leave the results in the same location. It is passed directly to xtrabackup's *xtrabackup --parallel* option. See the **xtrabackup** documentation for details

--password=PASSWORD
This option accepts a string argument specifying the password to use when connecting to the database. It is passed to the **mysql** child process without alteration. See **mysql --help** for details.

--port=PORT
This option accepts a string argument that specifies the port to use when connecting to the database server with TCP/IP. It is passed to the **mysql** child process. It is passed to the **mysql** child process without alteration. See **mysql --help** for details.

--rebuild-indexes
This option only has effect when used together with the *--apply-log* option and is passed directly to xtrabackup. When used, makes xtrabackup rebuild all secondary indexes after applying the log. This option is normally used to prepare compact backups. See the **xtrabackup** documentation for more information.

--rebuild-threads=NUMBER-OF-THREADS
This option only has effect when used together with the *innobackupex --apply-log* and *innobackupex --rebuild-indexes* option and is passed directly to xtrabackup. When used, xtrabackup processes tablespaces in parallel with the specified number of threads when rebuilding indexes. See the **xtrabackup** documentation for more information.

--redo-only
This option should be used when preparing the base full backup and when merging all incrementals except the last one. It is passed directly to xtrabackup's *xtrabackup --apply-log-only* option. This forces **xtrabackup** to skip the "rollback" phase and do a "redo" only. This is necessary if the backup will have incremental changes applied to it later. See the **xtrabackup** *documentation* for details.

--rsync
Uses the **rsync** utility to optimize local file transfers. When this option is specified, **innobackupex** uses **rsync** to copy all non-InnoDB files instead of spawning a separate **cp** for each file, which can be much faster for servers with a large number of databases or tables. This option cannot be used together with *innobackupex --stream*.

--safe-slave-backup
When specified, innobackupex will stop the slave SQL thread just before running `FLUSH TABLES WITH READ LOCK` and wait to start backup until `Slave_open_temp_tables` in `SHOW STATUS` is zero. If there are no open temporary tables, the backup will take place, otherwise the SQL thread will be started and stopped until there are no open temporary tables. The backup will fail if `Slave_open_temp_tables` does not become zero after *innobackupex --safe-slave-backup-timeout* seconds. The slave SQL thread will be restarted when the backup finishes.

--safe-slave-backup-timeout=SECONDS
How many seconds *innobackupex --safe-slave-backup* should wait for `Slave_open_temp_tables` to become zero. Defaults to 300 seconds.

--slave-info
This option is useful when backing up a replication slave server. It prints the binary log position and name of the master server. It also writes this information to the `xtrabackup_slave_info` file as a `CHANGE MASTER` command. A new slave for this master can be set up by starting a slave server on this backup and issuing a `CHANGE MASTER` command with the binary log position saved in the `xtrabackup_slave_info` file.

--socket
This option accepts a string argument that specifies the socket to use when connecting to the local database server with a UNIX domain socket. It is passed to the mysql child process without alteration. See **mysql --help** for details.

--stream=STREAMNAME
: This option accepts a string argument that specifies the format in which to do the streamed backup. The backup will be done to `STDOUT` in the specified format. Currently, supported formats are *tar* and *xbstream*. Uses *xbstream*, which is available in *Percona XtraBackup* distributions. If you specify a path after this option, it will be interpreted as the value of `tmpdir`

--tables-file=FILE
: This option accepts a string argument that specifies the file in which there are a list of names of the form `database.table`, one per line. The option is passed directly to **xtrabackup** 's *innobackupex --tables-file* option.

--throttle=#
: This option limits the number of chunks copied per second. The chunk size is *10 MB*. To limit the bandwidth to *10 MB/s*, set the option to *1*: *–throttle=1*.

  See also:

  More information about how to throttle a backup *Throttling Backups*

--tmpdir=DIRECTORY
: This option accepts a string argument that specifies the location where a temporary file will be stored. It may be used when *innobackupex --stream* is specified. For these options, the transaction log will first be stored to a temporary file, before streaming or copying to a remote host. This option specifies the location where that temporary file will be stored. If the option is not specified, the default is to use the value of `tmpdir` read from the server configuration. innobackupex is passing the tmpdir value specified in my.cnf as the –target-dir option to the xtrabackup binary. Both [mysqld] and [xtrabackup] groups are read from my.cnf. If there is tmpdir in both, then the value being used depends on the order of those group in my.cnf.

--use-memory=#
: This option accepts a string argument that specifies the amount of memory in bytes for **xtrabackup** to use for crash recovery while preparing a backup. Multiples are supported providing the unit (e.g. 1MB, 1M, 1GB, 1G). It is used only with the option *innobackupex --apply-log*. It is passed directly to xtrabackup's *xtrabackup --use-memory* option. See the **xtrabackup** documentation for details.

--user=USER
: This option accepts a string argument that specifies the user (i.e., the *MySQL* username used when connecting to the server) to login as, if that's not the current user. It is passed to the mysql child process without alteration. See **mysql --help** for details.

--version
: This option displays the **innobackupex** version and copyright notice and then exits.

The xtrabackup Binary

The **xtrabackup** binary is a compiled C program that is linked with the *InnoDB* libraries and the standard *MySQL* client libraries. The *InnoDB* libraries provide functionality necessary to apply a log to data files, and the *MySQL* client libraries provide command-line option parsing, configuration file parsing, and so on to give the binary a familiar look and feel.

The tool runs in either *xtrabackup --backup* or *xtrabackup --prepare* mode, corresponding to the two main functions it performs. There are several variations on these functions to accomplish different tasks, and there are two less commonly used modes, *xtrabackup --stats* and *xtrabackup --print-param*.

Other Types of Backups

Incremental Backups

Both **xtrabackup** and **innobackupex** tools supports incremental backups, which means that it can copy only the data that has changed since the last full backup. You can perform many incremental backups between each full backup, so you can set up a backup process such as a full backup once a week and an incremental backup every day, or full backups every day and incremental backups every hour.

Incremental backups work because each InnoDB page (usually 16kb in size) contains a log sequence number, or *LSN*. The *LSN* is the system version number for the entire database. Each page's *LSN* shows how recently it was changed. An incremental backup copies each page whose *LSN* is newer than the previous incremental or full backup's *LSN*. There are two algorithms in use to find the set of such pages to be copied. The first one, available with all the server types and versions, is to check the page *LSN* directly by reading all the data pages. The second one, available with *Percona Server for MySQL*, is to enable the changed page tracking feature on the server, which will note the pages as they are being changed. This information will be then written out in a compact separate so-called bitmap file. The **xtrabackup** binary will use that file to read only the data pages it needs for the incremental backup, potentially saving many read requests. The latter algorithm is enabled by default if the **xtrabackup** binary finds the bitmap file. It is possible to specify *xtrabackup --incremental-force-scan* to read all the pages even if the bitmap data is available.

Incremental backups do not actually compare the data files to the previous backup's data files. In fact, you can use *xtrabackup --incremental-lsn* to perform an incremental backup without even having the previous backup, if you know its *LSN*. Incremental backups simply read the pages and compare their *LSN* to the last backup's *LSN*. You still need a full backup to recover the incremental changes, however; without a full backup to act as a base, the incremental backups are useless.

Creating an Incremental Backup

To make an incremental backup, begin with a full backup as usual. The **xtrabackup** binary writes a file called `xtrabackup_checkpoints` into the backup's target directory. This file contains a line showing the `to_lsn`, which is the database's *LSN* at the end of the backup. *Create the full backup* with a command such as the following:

```
xtrabackup --backup --target-dir=/data/backups/base --datadir=/var/lib/mysql/
```

If you want a usable full backup, use *innobackupex* since *xtrabackup* itself won't copy table definitions, triggers, or anything else that's not .ibd.

If you look at the `xtrabackup_checkpoints` file, you should see some contents similar to the following:

```
backup_type = full-backuped
from_lsn = 0
to_lsn = 1291135
```

Now that you have a full backup, you can make an incremental backup based on it. Use a command such as the following:

```
xtrabackup --backup --target-dir=/data/backups/inc1 \
--incremental-basedir=/data/backups/base --datadir=/var/lib/mysql/
```

The `/data/backups/inc1/` directory should now contain delta files, such as `ibdata1.delta` and `test/table1.ibd.delta`. These represent the changes since the `LSN 1291135`. If you examine the `xtrabackup_checkpoints` file in this directory, you should see something similar to the following:

```
backup_type = incremental
from_lsn = 1291135
to_lsn = 1291340
```

The meaning should be self-evident. It's now possible to use this directory as the base for yet another incremental backup:

```
xtrabackup --backup --target-dir=/data/backups/inc2 \
--incremental-basedir=/data/backups/inc1 --datadir=/var/lib/mysql/
```

Preparing the Incremental Backups

The *xtrabackup --prepare* step for incremental backups is not the same as for normal backups. In normal backups, two types of operations are performed to make the database consistent: committed transactions are replayed from the log file against the data files, and uncommitted transactions are rolled back. You must skip the rollback of uncommitted transactions when preparing a backup, because transactions that were uncommitted at the time of your backup may be in progress, and it's likely that they will be committed in the next incremental backup. You should use the *xtrabackup --apply-log-only* option to prevent the rollback phase.

Warning: If you do not use the *xtrabackup --apply-log-only* option to prevent the rollback phase, then your incremental backups will be useless. After transactions have been rolled back, further incremental backups cannot be applied.

Beginning with the full backup you created, you can prepare it, and then apply the incremental differences to it. Recall that you have the following backups:

```
/data/backups/base
/data/backups/inc1
/data/backups/inc2
```

To prepare the base backup, you need to run *xtrabackup --prepare* as usual, but prevent the rollback phase:

```
xtrabackup --prepare --apply-log-only --target-dir=/data/backups/base
```

The output should end with some text such as the following:

```
101107 20:49:43  InnoDB: Shutdown completed; log sequence number 1291135
```

The log sequence number should match the `to_lsn` of the base backup, which you saw previously.

This backup is actually safe to *restore* as-is now, even though the rollback phase has been skipped. If you restore it and start *MySQL*, *InnoDB* will detect that the rollback phase was not performed, and it will do that in the background, as it usually does for a crash recovery upon start. It will notify you that the database was not shut down normally.

To apply the first incremental backup to the full backup, you should use the following command:

```
xtrabackup --prepare --apply-log-only --target-dir=/data/backups/base \
--incremental-dir=/data/backups/inc1
```

This applies the delta files to the files in `/data/backups/base`, which rolls them forward in time to the time of the incremental backup. It then applies the redo log as usual to the result. The final data is in `/data/backups/base`, not in the incremental directory. You should see some output such as the following:

```
incremental backup from 1291135 is enabled.
xtrabackup: cd to /data/backups/base/
xtrabackup: This target seems to be already prepared.
xtrabackup: xtrabackup_logfile detected: size=2097152, start_lsn=(1291340)
Applying /data/backups/inc1/ibdata1.delta ...
```

```
Applying /data/backups/inc1/test/table1.ibd.delta ...
.... snip
101107 20:56:30  InnoDB: Shutdown completed; log sequence number 1291340
```

Again, the *LSN* should match what you saw from your earlier inspection of the first incremental backup. If you restore the files from `/data/backups/base`, you should see the state of the database as of the first incremental backup.

Preparing the second incremental backup is a similar process: apply the deltas to the (modified) base backup, and you will roll its data forward in time to the point of the second incremental backup:

```
xtrabackup --prepare --target-dir=/data/backups/base \
--incremental-dir=/data/backups/inc2
```

Note: *`xtrabackup --apply-log-only`* should be used when merging all incrementals except the last one. That's why the previous line doesn't contain the *`xtrabackup --apply-log-only`* option. Even if the *`xtrabackup --apply-log-only`* was used on the last step, backup would still be consistent but in that case server would perform the rollback phase.

If you wish to avoid the notice that *InnoDB* was not shut down normally, when xoyou applied the desired deltas to the base backup, you can run *`xtrabackup --prepare`* again without disabling the rollback phase.

Partial Backups

`xtrabackup` supports taking partial backups when the *innodb_file_per_table* option is enabled. There are three ways to create partial backups:

1. matching the tables names with a regular expression
2. providing a list of table names in a file
3. providing a list of databases

> **Warning:** If any of the matched or listed tables is deleted during the backup, **`xtrabackup`** will fail.

For the purposes of this manual page, we will assume that there is a database named `test` which contains tables named `t1` and `t2`.

Using `xtrabackup --tables`

The first method involves the *`xtrabackup --tables`* option. The option's value is a regular expression that is matched against the fully qualified tablename, including the database name, in the form `databasename.tablename`.

To back up only tables in the `test` database, you can use the following command:

```
.. code-block:: bash
```

$ xtrabackup –backup –datadir=/var/lib/mysql –target-dir=/data/backups/ –tables="^test[.].*"

To back up only the table `test.t1`, you can use the following command:

```
.. code-block:: bash
```

$ xtrabackup –backup –datadir=/var/lib/mysql –target-dir=/data/backups/ –tables=”^test[.]t1”

Using `xtrabackup --tables-file`

xtrabackup --tables-file specifies a file that can contain multiple table names, one table name per line in the file. Only the tables named in the file will be backed up. Names are matched exactly, case-sensitive, with no pattern or regular expression matching. The table names must be fully qualified, in `databasename.tablename` format.

```
$ echo "mydatabase.mytable" > /tmp/tables.txt
$ xtrabackup --backup --tables-file=/tmp/tables.txt
```

Using `xtrabackup --databases` and `xtrabackup --databases-file`

xtrabackup --databases accepts a space-separated list of the databases and tables to backup in the format `databasename[.tablename]`. In addition to this list make sure to specify the `mysql`, `sys`, and `performance_schema` databases. These databases are required when restoring the databases using *xtrabackup --copy-back*.

```
$ xtrabackup --databases='mysql sys performance_schema ...'
```

xtrabackup --databases-file specifies a file that can contain multiple databases and tables in the `databasename[.tablename]` form, one element name per line in the file. Only named databases and tables will be backed up. Names are matched exactly, case-sensitive, with no pattern or regular expression matching.

Preparing the Backup

When you use the *xtrabackup --prepare* option on a partial backup, you will see warnings about tables that don't exist. This is because these tables exist in the data dictionary inside InnoDB, but the corresponding *.ibd* files don't exist. They were not copied into the backup directory. These tables will be removed from the data dictionary, and when you restore the backup and start InnoDB, they will no longer exist and will not cause any errors or warnings to be printed to the log file.

An example of the error message you will see during the prepare phase follows.

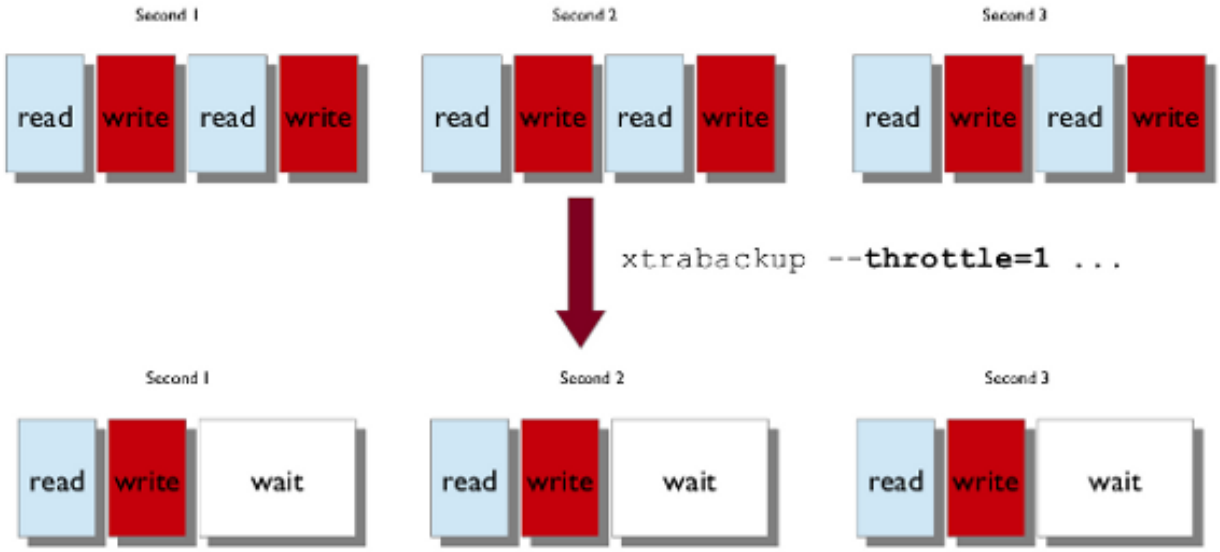
```
InnoDB: Reading tablespace information from the .ibd files...
101107 22:31:30  InnoDB: Error: table 'test1/t'
InnoDB: in InnoDB data dictionary has tablespace id 6,
InnoDB: but tablespace with that id or name does not exist. It will be removed from␣
↪data dictionary.
```

Advanced Features

Throttling Backups

Although **xtrabackup** does not block your database's operation, any backup can add load to the system being backed up. On systems that do not have much spare I/O capacity, it might be helpful to throttle the rate at which **xtrabackup** reads and writes data. You can do this with the *xtrabackup --throttle* option. This option limits the number of chunks copied per second. The chunk size is *10 MB*.

The image below shows how throttling works when *xtrabackup --throttle* is set to 1.



By default, there is no throttling, and **xtrabackup** reads and writes data as quickly as it can. If you set too strict of a limit on the IOPS, the backup might be so slow that it will never catch up with the transaction logs that InnoDB is writing, so the backup might never complete.

Scripting Backups With xtrabackup

The **xtrabackup** tool has several features to enable scripts to control it while they perform related tasks. The *innobackupex script* is one example, but **xtrabackup** is easy to control with your own command-line scripts too.

Suspending After Copying

In backup mode, **xtrabackup** normally copies the log files in a background thread, copies the data files in a foreground thread, and then stops the log copying thread and finishes. As long as that file exists, xtrabackup will continue to watch the log files and copy them into the `xtrabackup_logfile` in the target directory. When the file is removed, **xtrabackup** will finish copying the log file and exit.

This functionality is useful for coordinating the InnoDB data backups with other actions. Perhaps the most obvious is copying the table definitions (the `.frm` files) so that the backup can be restored. You can start **xtrabackup** in the background, wait for the `xtrabackup_suspended` file to be created, and then copy any other files you need to complete the backup. This is exactly what the *innobackupex* tool does (it also copies MyISAM data and other files).

Generating Meta-Data

It is a good idea for the backup to include all the information you need to restore the backup. The **xtrabackup** tool can print out the contents of a `my.cnf` file that are needed to restore the data and log files. *xtrabackup --print-param* prints out something like the following:

```
# This MySQL options file was generated by XtraBackup.
[mysqld]
datadir = /data/mysql/
innodb_data_home_dir = /data/innodb/
innodb_data_file_path = ibdata1:10M:autoextend
innodb_log_group_home_dir = /data/innodb-logs/
```

You can redirect this output into a file in the target directory of the backup.

Agreeing on the Source Directory

It's possible that the presence of a defaults file or other factors could cause **xtrabackup** to back up data from a different location than you expected. To prevent this, you can use *xtrabackup --print-param* to ask it where it will be copying data from. You can use the output to ensure that **xtrabackup** and your script are working on the same dataset.

Analyzing Table Statistics

The **xtrabackup** binary can analyze InnoDB data files in read-only mode to give statistics about them. To do this, you should use the *xtrabackup --stats* option. You can combine this with the *xtrabackup --tables* option to limit the files to examine. It also uses the *xtrabackup --use-memory* option.

You can perform the analysis on a running server, with some chance of errors due to the data being changed during analysis. Or, you can analyze a backup copy of the database. Either way, to use the statistics feature, you need a clean copy of the database including correctly sized log files, so you need to execute with *xtrabackup --prepare* twice to use this functionality on a backup.

The result of running on a backup might look like the following:

```
<INDEX STATISTICS>
  table: test/table1, index: PRIMARY, space id: 12, root page 3
  estimated statistics in dictionary:
    key vals: 25265338, leaf pages 497839, size pages 498304
  real statistics:
     level 2 pages: pages=1, data=5395 bytes, data/pages=32%
     level 1 pages: pages=415, data=6471907 bytes, data/pages=95%
        leaf pages: recs=25958413, pages=497839, data=7492026403 bytes, data/pages=91%
```

This can be interpreted as follows:

- The first line simply shows the table and index name and its internal identifiers. If you see an index named `GEN_CLUST_INDEX`, that is the table's clustered index, automatically created because you did not explicitly create a `PRIMARY KEY`.
- The estimated statistics in dictionary information is similar to the data that's gathered through `ANALYZE TABLE` inside of *InnoDB* to be stored as estimated cardinality statistics and passed to the query optimizer.
- The real statistics information is the result of scanning the data pages and computing exact information about the index.
- `The level <X> pages`: output means that the line shows information about pages at that level in the index tree. The larger `<X>` is, the farther it is from the leaf pages, which are level 0. The first line is the root page.
- The `leaf pages` output shows the leaf pages, of course. This is where the table's data is stored.
- The `external pages`: output (not shown) shows large external pages that hold values too long to fit in the row itself, such as long `BLOB` and `TEXT` values.
- The `recs` is the real number of records (rows) in leaf pages.
- The `pages` is the page count.
- The `data` is the total size of the data in the pages, in bytes.
- The `data/pages` is calculated as (`data` / (`pages` * `PAGE_SIZE`)) * 100%. It will never reach 100% because of space reserved for page headers and footers.

See also:

A more detailed example as a MySQL Performance blog post http://www.mysqlperformanceblog.com/2009/09/14/statistics-of-innodb-tables-and-indexes-available-in-xtrabackup/

Script to Format Output

The following script can be used to summarize and tabulate the output of the statistics information:

```
tabulate-xtrabackup-stats.pl

#!/usr/bin/env perl
use strict;
use warnings FATAL => 'all';
my $script_version = "0.1";

my $PG_SIZE = 16_384; # InnoDB defaults to 16k pages, change if needed.
my ($cur_idx, $cur_tbl);
my (%idx_stats, %tbl_stats);
my ($max_tbl_len, $max_idx_len) = (0, 0);
while ( my $line = <> ) {
   if ( my ($t, $i) = $line =~ m/table: (.*), index: (.*), space id:/ ) {
      $t =~ s!/!.!;
      $cur_tbl = $t;
      $cur_idx = $i;
      if ( length($i) > $max_idx_len ) {
         $max_idx_len = length($i);
      }
      if ( length($t) > $max_tbl_len ) {
         $max_tbl_len = length($t);
      }
   }
   elsif ( my ($kv, $lp, $sp) = $line =~ m/key vals: (\d+), \D*(\d+), \D*(\d+)/ ) {
      @{$idx_stats{$cur_tbl}->{$cur_idx}}{qw(est_kv est_lp est_sp)} = ($kv, $lp, $sp);
      $tbl_stats{$cur_tbl}->{est_kv} += $kv;
      $tbl_stats{$cur_tbl}->{est_lp} += $lp;
      $tbl_stats{$cur_tbl}->{est_sp} += $sp;
   }
   elsif ( my ($l, $pages, $bytes) = $line =~ m/(?:level (\d+)|leaf) pages:.
↪*pages=(\d+), data=(\d+) bytes/ ) {
      $l ||= 0;
      $idx_stats{$cur_tbl}->{$cur_idx}->{real_pages} += $pages;
      $idx_stats{$cur_tbl}->{$cur_idx}->{real_bytes} += $bytes;
      $tbl_stats{$cur_tbl}->{real_pages} += $pages;
      $tbl_stats{$cur_tbl}->{real_bytes} += $bytes;
   }
}

my $hdr_fmt = "%${max_tbl_len}s %${max_idx_len}s %9s %10s %10s\n";
my @headers = qw(TABLE INDEX TOT_PAGES FREE_PAGES PCT_FULL);
printf $hdr_fmt, @headers;

my $row_fmt = "%${max_tbl_len}s %${max_idx_len}s %9d %10d %9.1f%%\n";
foreach my $t ( sort keys %tbl_stats ) {
   my $tbl = $tbl_stats{$t};
   printf $row_fmt, $t, "", $tbl->{est_sp}, $tbl->{est_sp} - $tbl->{real_pages},
      $tbl->{real_bytes} / ($tbl->{real_pages} * $PG_SIZE) * 100;
   foreach my $i ( sort keys %{$idx_stats{$t}} ) {
```

```
      my $idx = $idx_stats{$t}->{$i};
      printf $row_fmt, $t, $i, $idx->{est_sp}, $idx->{est_sp} - $idx->{real_pages},
         $idx->{real_bytes} / ($idx->{real_pages} * $PG_SIZE) * 100;
   }
}
```

Sample Script Output

The output of the above Perl script, when run against the sample shown in the previously mentioned blog post, will appear as follows:

```
          TABLE           INDEX TOT_PAGES FREE_PAGES   PCT_FULL
art.link_out104                    832383      38561      86.8%
art.link_out104         PRIMARY    498304         49      91.9%
art.link_out104       domain_id     49600       6230      76.9%
art.link_out104     domain_id_2     26495       3339      89.1%
art.link_out104 from_message_id     28160        142      96.3%
art.link_out104    from_site_id     38848       4874      79.4%
art.link_out104   revert_domain    153984      19276      71.4%
art.link_out104    site_message     36992       4651      83.4%
```

The columns are the table and index, followed by the total number of pages in that index, the number of pages not actually occupied by data, and the number of bytes of real data as a percentage of the total size of the pages of real data. The first line in the above output, in which the `INDEX` column is empty, is a summary of the entire table.

Working with Binary Logs

The `xtrabackup` binary integrates with information that *InnoDB* stores in its transaction log about the corresponding binary log position for committed transactions. This enables it to print out the binary log position to which a backup corresponds, so you can use it to set up new replication slaves or perform point-in-time recovery.

Finding the Binary Log Position

You can find the binary log position corresponding to a backup once the backup has been prepared. This can be done by either running *`xtrabackup --prepare`* or *`innobackupex --apply-log`*. If your backup is from a server with binary logging enabled, **`xtrabackup`** will create a file named `xtrabackup_binlog_info` in the target directory. This file contains the binary log file name and position of the exact point in the binary log to which the prepared backup corresponds.

You will also see output similar to the following during the `prepare` stage:

```
InnoDB: Last MySQL binlog file position 0 3252710, file name ./mysql-bin.000001
... snip ...
[notice (again)]
  If you use binary log and don't use any hack of group commit,
  the binary log position seems to be:
InnoDB: Last MySQL binlog file position 0 3252710, file name ./mysql-bin.000001
```

This output can also be found in the `xtrabackup_binlog_pos_innodb` file, but **it is only correct** when no other than *XtraDB* or *InnoDB* are used as storage engines.

If other storage engines are used (i.e. *MyISAM*), you should use the `xtrabackup_binlog_info` file to retrieve the position.

The message about hacking group commit refers to an early implementation of emulated group commit in *Percona Server for MySQL*.

Point-In-Time Recovery

To perform a point-in-time recovery from an `xtrabackup` backup, you should prepare and restore the backup, and then replay binary logs from the point shown in the `xtrabackup_binlog_info` file.

A more detailed procedure is found *here* (with **`innobackupex`**).

Setting Up a New Replication Slave

To set up a new replica, you should prepare the backup, and restore it to the data directory of your new replication slave. Then in your `CHANGE MASTER TO` command, use the binary log filename and position shown in the `xtrabackup_binlog_info` file to start replication.

A more detailed procedure is found in *How to setup a slave for replication in 6 simple steps with Percona XtraBackup*.

Restoring Individual Tables

In server versions prior to 5.6, it is not possible to copy tables between servers by copying the files, even with *innodb_file_per_table*. However, with *Percona XtraBackup*, you can export individual tables from any *InnoDB* database, and import them into *Percona Server for MySQL* with *XtraDB* or *MySQL* 5.6. (The source doesn't have to be *XtraDB* or or *MySQL* 5.6, but the destination does.) This only works on individual *.ibd* files, and cannot export a table that is not contained in its own *.ibd* file.

Let's see how to export and import the following table:

```
CREATE TABLE export_test (
  a int(11) DEFAULT NULL
) ENGINE=InnoDB DEFAULT CHARSET=latin1;
```

Note: If you're running *Percona Server for MySQL* version older than 5.5.10-20.1, variable innodb_expand_import should be used instead of innodb_import_table_from_xtrabackup.

Exporting the Table

This table should have been created in *innodb_file_per_table* mode, so after taking a backup as usual with *`xtrabackup --backup`*, the *.ibd* file should exist in the target directory:

```
$ find /data/backups/mysql/ -name export_test.*
/data/backups/mysql/test/export_test.ibd
```

when you prepare the backup, add the extra parameter *`xtrabackup --export`* to the command. Here is an example:

```
$ xtrabackup --prepare --export --target-dir=/data/backups/mysql/
```

Note: If you're trying to restore *encrypted InnoDB tablespace* table you'll need to specify the keyring file as well:

```
xtrabackup --prepare --export --target-dir=/tmp/table \
--keyring-file-data=/var/lib/mysql-keyring/keyring
```

Now you should see a *.exp* file in the target directory:

```
$ find /data/backups/mysql/ -name export_test.*
/data/backups/mysql/test/export_test.exp
/data/backups/mysql/test/export_test.ibd
/data/backups/mysql/test/export_test.cfg
```

These three files are all you need to import the table into a server running *Percona Server for MySQL* with *XtraDB* or *MySQL* 5.7. In case server is using InnoDB Tablespace Encryption additional `.cfp` file be listed for encrypted tables.

Note: *MySQL* uses `.cfg` file which contains *InnoDB* dictionary dump in special format. This format is different from the `.exp`` one which is used in *XtraDB* for the same purpose. Strictly speaking, a `.cfg`` file is not required to import a tablespace to *MySQL* 5.7 or *Percona Server for MySQL* 5.7. A tablespace will be imported successfully even if it is from another server, but *InnoDB* will do schema validation if the corresponding `.cfg` file is present in the same directory.

Importing the Table

On the destination server running *Percona Server for MySQL* with *XtraDB* and innodb_import_table_from_xtrabackup option enabled, or *MySQL* 5.6, create a table with the same structure, and then perform the following steps:

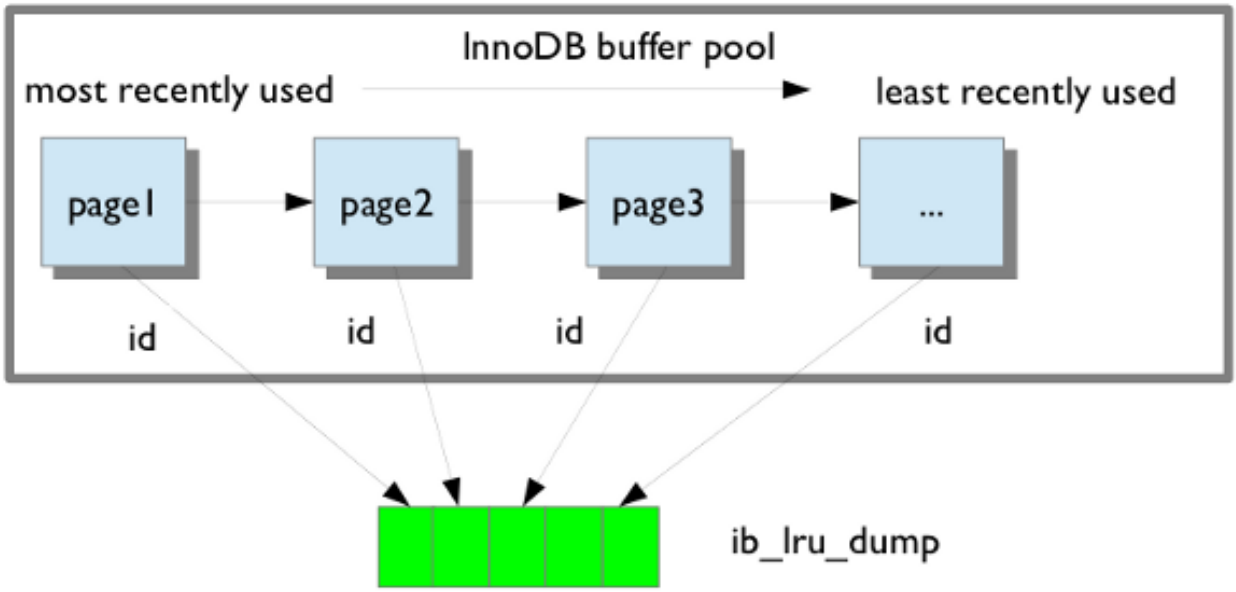
- Execute `ALTER TABLE test.export_test DISCARD TABLESPACE;`
  - If you see the following message, then you must enable *innodb_file_per_table* and create the table again: `ERROR 1030 (HY000): Got error -1 from storage engine`
- Copy the exported files to the `test/` subdirectory of the destination server's data directory
- Execute `ALTER TABLE test.export_test IMPORT TABLESPACE;`

The table should now be imported, and you should be able to `SELECT` from it and see the imported data.

Note: Persistent statistics for imported tablespace will be empty until you run the `ANALYZE TABLE` on the imported table. They will be empty because they are stored in the system tables `mysql.innodb_table_stats` and `mysql.innodb_index_stats` and they aren't updated by server during the import. This is due to upstream bug #72368.

LRU dump backup

This feature reduces the warm up time by restoring buffer pool state from `ib_lru_dump` file after restart. *Percona XtraBackup* discovers `ib_lru_dump` and backs it up automatically.



If the buffer restore option is enabled in `my.cnf` buffer pool will be in the warm state after backup is restored. To enable this set the variable innodb_buffer_pool_restore_at_startup =1 in Percona Server 5.5 or innodb_auto_lru_dump =1 in Percona Server 5.1.

Implementation

Implementation Details

This page contains notes on various internal aspects of the **`xtrabackup`** tool's operation.

File Permissions

`xtrabackup` opens the source data files in read-write mode, although it does not modify the files. This means that you must run **`xtrabackup`** as a user who has permission to write the data files. The reason for opening the files in read-write mode is that **`xtrabackup`** uses the embedded *InnoDB* libraries to open and read the files, and *InnoDB* opens them in read-write mode because it normally assumes it is going to write to them.

Tuning the OS Buffers

Because **`xtrabackup`** reads large amounts of data from the filesystem, it uses `posix_fadvise()` where possible, to instruct the operating system not to try to cache the blocks it reads from disk. Without this hint, the operating system would prefer to cache the blocks, assuming that `xtrabackup` is likely to need them again, which is not the case. Caching such large files can place pressure on the operating system's virtual memory and cause other processes, such as the database server, to be swapped out. The `xtrabackup` tool avoids this with the following hint on both the source and destination files:

```
posix_fadvise(file, 0, 0, POSIX_FADV_DONTNEED)
```

In addition, xtrabackup asks the operating system to perform more aggressive read-ahead optimizations on the source files:

```
posix_fadvise(file, 0, 0, POSIX_FADV_SEQUENTIAL)
```

Copying Data Files

When copying the data files to the target directory, **xtrabackup** reads and writes 1MB of data at a time. This is not configurable. When copying the log file, **xtrabackup** reads and writes 512 bytes at a time. This is also not possible to configure, and matches InnoDB's behavior (workaround exists in *Percona Server for MySQL* because it has an option to tune `innodb_log_block_size` for *XtraDB*, and in that case *Percona XtraBackup* will match the tuning).

After reading from the files, `xtrabackup` iterates over the 1MB buffer a page at a time, and checks for page corruption on each page with InnoDB's `buf_page_is_corrupted()` function. If the page is corrupt, it re-reads and retries up to 10 times for each page. It skips this check on the doublewrite buffer.

`xtrabackup` Exit Codes

The **xtrabackup** binary exits with the traditional success value of 0 after a backup when no error occurs. If an error occurs during the backup, the exit value is 1.

In certain cases, the exit value can be something other than 0 or 1, due to the command-line option code included from the *MySQL* libraries. An unknown command-line option, for example, will cause an exit code of 255.

References

The `xtrabackup` Option Reference

This page documents all of the command-line options for the **xtrabackup** binary.

Options

--apply-log-only
: This option causes only the redo stage to be performed when preparing a backup. It is very important for incremental backups.

--backup
: Make a backup and place it in *`xtrabackup --target-dir`*. See *Creating a backup*.

--binlog-info
: This option controls how *Percona XtraBackup* should retrieve server's binary log coordinates corresponding to the backup. Possible values are `OFF`, `ON`, `LOCKLESS` and `AUTO`. See the *Percona XtraBackup* *Lockless binary log information* manual page for more information.

--check-privileges
: This option checks if *Percona XtraBackup* has all required privileges. If a missing privilege is required for the current operation, it will terminate and print out an error message. If a missing privilege is not required for the current operation, but may be necessary for some other XtraBackup operation, the process is not aborted and a warning is printed.

```
xtrabackup: Error: missing required privilege LOCK TABLES on *.*
xtrabackup: Warning: missing required privilege REPLICATION CLIENT on *.*
```

--close-files
Do not keep files opened. When **xtrabackup** opens tablespace it normally doesn't close its file handle in order to handle the DDL operations correctly. However, if the number of tablespaces is really huge and can not fit into any limit, there is an option to close file handles once they are no longer accessed. *Percona XtraBackup* can produce inconsistent backups with this option enabled. Use at your own risk.

--compress
This option tells **xtrabackup** to compress all output data, including the transaction log file and meta data files, using the specified compression algorithm. The only currently supported algorithm is `quicklz`. The resulting files have the qpress archive format, i.e. every `*.qp` file produced by xtrabackup is essentially a one-file qpress archive and can be extracted and uncompressed by the qpress file archiver.

--compress-chunk-size=#
Size of working buffer(s) for compression threads in bytes. The default value is 64K.

--compress-threads=#
This option specifies the number of worker threads used by **xtrabackup** for parallel data compression. This option defaults to `1`. Parallel compression (:option:‘ xtrabackup –compress-threads‘) can be used together with parallel file copying (*xtrabackup --parallel*). For example, `--parallel=4 --compress --compress-threads=2` will create 4 I/O threads that will read the data and pipe it to 2 compression threads.

--copy-back
Copy all the files in a previously made backup from the backup directory to their original locations. This option will not copy over existing files unless *xtrabackup --force-non-empty-directories* option is specified.

--databases=#
This option specifies the list of databases and tables that should be backed up. The option accepts the list of the form `"databasename1[.table_name1] databasename2[.table_name2] . . ."`.

--databases-exclude=name
Excluding databases based on name, Operates the same way as *xtrabackup --databases*, but matched names are excluded from backup. Note that this option has a higher priority than *xtrabackup --databases*.

--databases-file=#
This option specifies the path to the file containing the list of databases and tables that should be backed up. The file can contain the list elements of the form `databasename1[.table_name1]`, one element per line.

--datadir=DIRECTORY
The source directory for the backup. This should be the same as the datadir for your *MySQL* server, so it should be read from `my.cnf` if that exists; otherwise you must specify it on the command line.

When combined with the *xtrabackup --copy-back* or *xtrabackup --move-back* option, *xtrabackup --datadir* refers to the destination directory.

Once connected to the server, in order to perform a backup you will need `READ` and `EXECUTE` permissions at a filesystem level in the server's *datadir*.

--decompress
Decompresses all files with the `.qp` extension in a backup previously made with the *xtrabackup --compress* option. The *xtrabackup --parallel* option will allow multiple files to be decrypted simultaneously. In order to decompress, the qpress utility MUST be installed and accessible within the path. *Percona XtraBackup* doesn't automatically remove the compressed files. In order to clean up the backup directory users should use *xtrabackup --remove-original* option.

--decrypt=ENCRYPTION-ALGORITHM
Decrypts all files with the `.xbcrypt` extension in a backup previously made with *xtrabackup*

--encrypt option. The *xtrabackup --parallel* option will allow multiple files to be decrypted simultaneously. *Percona XtraBackup* doesn't automatically remove the encrypted files. In order to clean up the backup directory users should use *xtrabackup --remove-original* option.

--defaults-extra-file=[MY.CNF]
Read this file after the global files are read. Must be given as the first option on the command-line.

--defaults-file=[MY.CNF]
Only read default options from the given file. Must be given as the first option on the command-line. Must be a real file; it cannot be a symbolic link.

--defaults-group=GROUP-NAME
This option is to set the group which should be read from the configuration file. This is used by **innobackupex** if you use the *xtrabackup --defaults-group* option. It is needed for `mysqld_multi` deployments.

--dump-innodb-buffer-pool
This option controls whether or not a new dump of buffer pool content should be done.

With `--dump-innodb-buffer-pool`, **xtrabackup** makes a request to the server to start the buffer pool dump (it takes some time to complete and is done in background) at the beginning of a backup provided the status variable `innodb_buffer_pool_dump_status` reports that the dump has been completed.

```
$ xtrabackup --backup --dump-innodb-buffer-pool --target-dir=/home/user/backup
```

By default, this option is set to *OFF*.

If `innodb_buffer_pool_dump_status` reports that there is running dump of buffer pool, **xtrabackup** waits for the dump to complete using the value of *--dump-innodb-buffer-pool-timeout*

The file `ib_buffer_pool` stores tablespace ID and page ID data used to warm up the buffer pool sooner.

See also:

***MySQL* Documentation: Saving and Restoring the Buffer Pool State** https://dev.mysql.com/doc/refman/5.7/en/innodb-preload-buffer-pool.html

--dump-innodb-buffer-pool-timeout
This option contains the number of seconds that **xtrabackup** should monitor the value of `innodb_buffer_pool_dump_status` to determine if buffer pool dump has completed.

This option is used in combination with *--dump-innodb-buffer-pool*. By default, it is set to *10* seconds.

--dump-innodb-buffer-pool-pct
This option contains the percentage of the most recently used buffer pool pages to dump.

This option is effective if *--dump-innodb-buffer-pool* option is set to *ON*. If this option contains a value, **xtrabackup** sets the *MySQL* system variable `innodb_buffer_pool_dump_pct`. As soon as the buffer pool dump completes or it is stopped (see *--dump-innodb-buffer-pool-timeout*), the value of the *MySQL* system variable is restored.

See also:

Changing the timeout for buffer pool dump *--dump-innodb-buffer-pool-timeout*

***MySQL* Documentation: innodb_buffer_pool_dump_pct system variable** https://dev.mysql.com/doc/refman/8.0/en/innodb-parameters.html#sysvar_innodb_buffer_pool_dump_pct

--encrypt=ENCRYPTION_ALGORITHM
This option instructs xtrabackup to encrypt backup copies of InnoDB data files using the algorithm specified in the ENCRYPTION_ALGORITHM. It is passed directly to the xtrabackup child process. See the **xtrabackup** *documentation* for more details.

--encrypt-key=ENCRYPTION_KEY
This option instructs xtrabackup to use the given ENCRYPTION_KEY when using the *xtrabackup --encrypt* option. It is passed directly to the xtrabackup child process. See the **xtrabackup** *documentation* for more details.

--encrypt-key-file=ENCRYPTION_KEY_FILE
This option instructs xtrabackup to use the encryption key stored in the given ENCRYPTION_KEY_FILE when using the *xtrabackup --encrypt* option. It is passed directly to the xtrabackup child process. See the **xtrabackup** *documentation* for more details.

--encrypt-threads=#
This option specifies the number of worker threads that will be used for parallel encryption/decryption. See the **xtrabackup** *documentation* for more details.

--encrypt-chunk-size=#
This option specifies the size of the internal working buffer for each encryption thread, measured in bytes. It is passed directly to the xtrabackup child process. See the **xtrabackup** *documentation* for more details.

--export
Create files necessary for exporting tables. See *Restoring Individual Tables*.

--extra-lsndir=DIRECTORY
(for –backup): save an extra copy of the xtrabackup_checkpoints and xtrabackup_info files in this directory.

--force-non-empty-directories
When specified, it makes :option‘xtrabackup –copy-back‘ and *xtrabackup --move-back* option transfer files to non-empty directories. No existing files will be overwritten. If files that need to be copied/moved from the backup directory already exist in the destination directory, it will still fail with an error.

--ftwrl-wait-timeout=SECONDS
This option specifies time in seconds that xtrabackup should wait for queries that would block FLUSH TABLES WITH READ LOCK before running it. If there are still such queries when the timeout expires, xtrabackup terminates with an error. Default is 0, in which case it does not wait for queries to complete and starts FLUSH TABLES WITH READ LOCK immediately. Where supported (Percona Server 5.6+) xtrabackup will automatically use Backup Locks as a lightweight alternative to FLUSH TABLES WITH READ LOCK to copy non-InnoDB data to avoid blocking DML queries that modify InnoDB tables.

--ftwrl-wait-threshold=SECONDS
This option specifies the query run time threshold which is used by xtrabackup to detect long-running queries with a non-zero value of *xtrabackup --ftwrl-wait-timeout*. FLUSH TABLES WITH READ LOCK is not started until such long-running queries exist. This option has no effect if *xtrabackup --ftwrl-wait-timeout* is 0. Default value is 60 seconds. Where supported (Percona Server 5.6+) xtrabackup will automatically use Backup Locks as a lightweight alternative to FLUSH TABLES WITH READ LOCK to copy non-InnoDB data to avoid blocking DML queries that modify InnoDB tables.

--ftwrl-wait-query-type=all|update
This option specifies which types of queries are allowed to complete before xtrabackup will issue the global lock. Default is all.

--galera-info
This options creates the xtrabackup_galera_info file which contains the local node state at the time of the backup. Option should be used when performing the backup of *Percona XtraDB Cluster*. It has no effect when backup locks are used to create the backup.

--history=name
This option enables the tracking of the backup history in the PERCONA_SCHEMA.xtrabackup_history table. An optional history series name may be specified that will be placed with the history record for the backup being taken.

--incremental-basedir=DIRECTORY
When creating an incremental backup, this is the directory containing the full backup that is the base dataset for the incremental backups.

--incremental-dir=DIRECTORY
When preparing an incremental backup, this is the directory where the incremental backup is combined with the full backup to make a new full backup.

--incremental-force-scan
When creating an incremental backup, force a full scan of the data pages in the instance being backuped even if the complete changed page bitmap data is available.

--incremental-history-name=name
This option specifies the name of the backup series stored in the *PERCONA_SCHEMA.xtrabackup_history* history record to base an incremental backup on. **xtrabackup** searches the history table for the most recent (highest innodb_to_lsn), successful backup in the series and take the to_lsn value to use as the starting lsn for the incremental backup. This will be mutually exclusive with *xtrabackup --incremental-history-uuid*, *xtrabackup --incremental-basedir* and *xtrabackup --incremental-lsn*. If no valid *LSN* can be found (no series by that name, no successful backups by that name) **xtrabackup** returns an error. It is used with the `xtrabackup --incremental` option.

--incremental-history-uuid=UUID
This option specifies the *UUID* of the specific history record stored in the *PERCONA_SCHEMA.xtrabackup_history* to base an incremental backup on. *xtrabackup --incremental-history-name*, *xtrabackup --incremental-basedir* and *xtrabackup --incremental-lsn*. If no valid *LSN* is found (no success record with that *UUID*) **xtrabackup** returns an error. This option is used with the `xtrabackup --incremental` option.

--incremental-lsn=LSN
When creating an incremental backup, you can specify the log sequence number (*LSN*) instead of specifying *xtrabackup --incremental-basedir*. For databases created in 5.1 and later, specify the *LSN* as a single 64-bit integer. **ATTENTION**: If a wrong LSN value is specified (a user error which *Percona XtraBackup* is unable to detect), the backup will be unusable. Be careful!

--innodb-log-arch-dir=DIRECTORY
This option is used to specify the directory containing the archived logs. It can only be used with the *xtrabackup --prepare* option.

--innodb-miscellaneous
There is a large group of InnoDB options that are normally read from the `my.cnf` configuration file, so that **xtrabackup** boots up its embedded InnoDB in the same configuration as your current server. You normally do not need to specify these explicitly. These options have the same behavior that they have in InnoDB or XtraDB. They are as follows:

```
--innodb-adaptive-hash-index
--innodb-additional-mem-pool-size
--innodb-autoextend-increment
--innodb-buffer-pool-size
--innodb-checksums
--innodb-data-file-path
--innodb-data-home-dir
--innodb-doublewrite-file
--innodb-doublewrite
--innodb-extra-undoslots
--innodb-fast-checksum
--innodb-file-io-threads
--innodb-file-per-table
--innodb-flush-log-at-trx-commit
--innodb-flush-method
```

```
--innodb-force-recovery
--innodb-io-capacity
--innodb-lock-wait-timeout
--innodb-log-buffer-size
--innodb-log-files-in-group
--innodb-log-file-size
--innodb-log-group-home-dir
--innodb-max-dirty-pages-pct
--innodb-open-files
--innodb-page-size
--innodb-read-io-threads
--innodb-write-io-threads
```

--keyring-file-data=FILENAME
The path to the keyring file. Combine this option with *xtrabackup --xtrabackup-plugin-dir*.

--lock-ddl
Issue `LOCK TABLES FOR BACKUP` if it is supported by server at the beginning of the backup to block all DDL operations.

--lock-ddl-per-table
Lock DDL for each table before xtrabackup starts to copy it and until the backup is completed.

--lock-ddl-timeout
If `LOCK TABLES FOR BACKUP` does not return within given timeout, abort the backup.

--log-copy-interval=#
This option specifies time interval between checks done by log copying thread in milliseconds (default is 1 second).

--move-back
Move all the files in a previously made backup from the backup directory to their original locations. As this option removes backup files, it must be used with caution.

--no-defaults
Don't read default options from any option file. Must be given as the first option on the command-line.

--no-version-check
This option disables the version check. If you do not pass this option, the automatic version check is enabled implicitly when **xtrabackup** runs in the `--backup` mode. To disable the version check, you should pass explicitly the `--no-version-check` option when envoking **xtrabackup**.

When the automatic version check is enabled, **xtrabackup** performs a version check against the server on the backup stage after creating a server connection. **xtrabackup** sends the following information to the server:

- MySQL flavour and version
- Operating system name
- Percona Toolkit version
- Perl version

Each piece of information has a unique identifier. This is a MD5 hash value that Percona Toolkit uses to obtain statistics about how it is used. This is a random UUID; no client information is either collected or stored.

--parallel=#
This option specifies the number of threads to use to copy multiple data files concurrently when creating a backup. The default value is 1 (i.e., no concurrent transfer). In *Percona XtraBackup* 2.3.10 and newer, this option can be used with *xtrabackup --copy-back* option to copy the user data files in parallel (redo logs and system tablespaces are copied in the main thread).

--password=PASSWORD
This option specifies the password to use when connecting to the database. It accepts a string argument. See mysql –help for details.

--prepare
Makes **xtrabackup** perform recovery on a backup created with *xtrabackup --backup*, so that it is ready to use. See *preparing a backup*.

--print-defaults
Print the program argument list and exit. Must be given as the first option on the command-line.

--print-param
Makes **xtrabackup** print out parameters that can be used for copying the data files back to their original locations to restore them. See *Scripting Backups With xtrabackup*.

--reencrypt-for-server-id=<new_server_id>
Use this option to start the server instance with different server_id from the one the encrypted backup was taken from, like a replication slave or a galera node. When this option is used, xtrabackup will, as a prepare step, generate a new master key with ID based on the new server_id, store it into keyring file and re-encrypt the tablespace keys inside of tablespace headers. Option should be passed for *--prepare* (final step).

--remove-original
Implemented in *Percona XtraBackup* 2.4.6, this option when specified will remove `.qp`, `.xbcrypt` and `.qp.xbcrypt` files after decryption and decompression.

--safe-slave-backup
When specified, xtrabackup will stop the slave SQL thread just before running `FLUSH TABLES WITH READ LOCK` and wait to start backup until `Slave_open_temp_tables` in `SHOW STATUS` is zero. If there are no open temporary tables, the backup will take place, otherwise the SQL thread will be started and stopped until there are no open temporary tables. The backup will fail if `Slave_open_temp_tables` does not become zero after *xtrabackup --safe-slave-backup-timeout* seconds. The slave SQL thread will be restarted when the backup finishes. This option is implemented in order to deal with replicating temporary tables and isn't neccessary with Row-Based-Replication.

--safe-slave-backup-timeout=SECONDS
How many seconds *xtrabackup --safe-slave-backup* should wait for `Slave_open_temp_tables` to become zero. Defaults to 300 seconds.

--secure-auth
Refuse client connecting to server if it uses old (pre-4.1.1) protocol. (Enabled by default; use –skip-secure-auth to disable.)

--server-id=#
The server instance being backed up.

--slave-info
This option is useful when backing up a replication slave server. It prints the binary log position of the master server. It also writes the binary log coordinates to the `xtrabackup_slave_info` file as a `CHANGE MASTER` command. A new slave for this master can be set up by starting a slave server on this backup and issuing a `CHANGE MASTER` command with the binary log position saved in the `xtrabackup_slave_info` file.

--ssl
Enable secure connection. More information can be found in –ssl MySQL server documentation.

--ssl-ca
Path of the file which contains list of trusted SSL CAs. More information can be found in –ssl-ca MySQL server documentation.

--ssl-capath
Directory path that contains trusted SSL CA certificates in PEM format. More information can be found in –ssl-capath MySQL server documentation.

--ssl-cert
Path of the file which contains X509 certificate in PEM format. More information can be found in –ssl-cert MySQL server documentation.

--ssl-cipher
List of permitted ciphers to use for connection encryption. More information can be found in –ssl-cipher MySQL server documentation.

--ssl-crl
Path of the file that contains certificate revocation lists. More information can be found in –ssl-crl MySQL server documentation.

--ssl-crlpath
Path of directory that contains certificate revocation list files. More information can be found in –ssl-crlpath MySQL server documentation.

--ssl-key
Path of file that contains X509 key in PEM format. More information can be found in –ssl-key MySQL server documentation.

--ssl-mode
Security state of connection to server. More information can be found in –ssl-mode MySQL server documentation.

--ssl-verify-server-cert
Verify server certificate Common Name value against host name used when connecting to server. More information can be found in –ssl-verify-server-cert MySQL server documentation.

--stats
Causes **xtrabackup** to scan the specified data files and print out index statistics.

--stream=name
Stream all backup files to the standard output in the specified format. Currently supported formats are `xbstream` and `tar`.

--tables=name
A regular expression against which the full tablename, in `databasename.tablename` format, is matched. If the name matches, the table is backed up. See *partial backups*.

--tables-exclude=name
Filtering by regexp for table names. Operates the same way as *`xtrabackup --tables`*, but matched names are excluded from backup. Note that this option has a higher priority than *`xtrabackup --tables`*.

--tables-file=name
A file containing one table name per line, in databasename.tablename format. The backup will be limited to the specified tables. See *Scripting Backups With xtrabackup*.

--target-dir=DIRECTORY
This option specifies the destination directory for the backup. If the directory does not exist, **xtrabackup** creates it. If the directory does exist and is empty, **xtrabackup** will succeed. **xtrabackup** will not overwrite existing files, however; it will fail with operating system error 17, `file exists`.

If this option is a relative path, it is interpreted as being relative to the current working directory from which **xtrabackup** is executed.

In order to perform a backup, you need `READ`, `WRITE`, and `EXECUTE` permissions at a filesystem level for the directory that you supply as the value of *`--target-dir`*.

--throttle=#
This option limits the number of chunks copied per second. The chunk size is *10 MB*. To limit the bandwidth to *10 MB/s*, set the option to *1*: *–throttle=1*.

See also:

More information about how to throttle a backup *Throttling Backups*

--tmpdir=name
This option is currently not used for anything except printing out the correct tmpdir parameter when *xtrabackup --print-param* is used.

--to-archived-lsn=LSN
This option is used to specify the LSN to which the logs should be applied when backups are being prepared. It can only be used with the *xtrabackup --prepare* option.

--transition-key
This option is used to enable processing the backup without accessing the keyring vault server. In this case, **xtrabackup** derives the AES encryption key from the specified passphrase and uses it to encrypt tablespace keys of tablespaces being backed up.

If *--transition-key* does not have any value, **xtrabackup** will ask for it. The same passphrase should be specified for the *xtrabackup --prepare* command.

--use-memory=#
This option affects how much memory is allocated for preparing a backup with *xtrabackup --prepare*, or analyzing statistics with *xtrabackup --stats*. Its purpose is similar to *innodb_buffer_pool_size*. It does not do the same thing as the similarly named option in Oracle's InnoDB Hot Backup tool. The default value is 100MB, and if you have enough available memory, 1GB to 2GB is a good recommended value. Multiples are supported providing the unit (e.g. 1MB, 1M, 1GB, 1G).

--user=USERNAME
This option specifies the MySQL username used when connecting to the server, if that's not the current user. The option accepts a string argument. See mysql –help for details.

--version
This option prints **xtrabackup** version and exits.

--xtrabackup-plugin-dir=DIRNAME
The absolute path to the directory that contains the keyring plugin.

See also:

***Percona Server for MySQL* Documentation: keyring_vault plugin with Data at Rest Encryption**
https://www.percona.com/doc/percona-server/LATEST/management/data_at_rest_encryption.html#keyring-vault-plugin

***MySQL* Documentation: Using the keyring_file File-Based Plugin** https://dev.mysql.com/doc/refman/5.7/en/keyring-file-plugin.html

The xbstream binary

To support simultaneous compression and streaming, a new custom streaming format called xbstream was introduced to *Percona XtraBackup* in addition to the TAR format. That was required to overcome some limitations of traditional archive formats such as tar, cpio and others which did not allow streaming dynamically generated files, for example dynamically compressed files. Other advantages of xbstream over traditional streaming/archive format include ability

to stream multiple files concurrently (so it is possible to use streaming in the xbstream format together with the –parallel option) and more compact data storage.

This utility has a tar-like interface:

- with the `-x` option it extracts files from the stream read from its standard input to the current directory unless specified otherwise with the `-c` option. Support for parallel extraction with the `--parallel` option has been implemented in *Percona XtraBackup* 2.4.7.
- with the `-c` option it streams files specified on the command line to its standard output.
- with the `--decrypt=ALGO` option specified xbstream will automatically decrypt encrypted files when extracting input stream. Supported values for this option are: `AES128`, `AES192`, and `AES256`. Either `--encrypt-key` or `--encrypt-key-file` options must be specified to provide encryption key, but not both. This option has been implemented in *Percona XtraBackup* 2.4.7.
- with the `--encrypt-threads` option you can specify the number of threads for parallel data encryption. The default value is `1`. This option has been implemented in *Percona XtraBackup* 2.4.7.
- the `--encrypt-key` option is used to specify the encryption key that will be used. It can't be used with `--encrypt-key-file` option because they are mutually exclusive. This option has been implemented in *Percona XtraBackup* 2.4.7.
- the `--encrypt-key-file` option is used to specify the file that contains the encryption key. It can't be used with `--encrypt-key` option. because they are mutually exclusive. This option has been implemented in *Percona XtraBackup* 2.4.7.

The utility also tries to minimize its impact on the OS page cache by using the appropriate `posix_fadvise()` calls when available.

When compression is enabled with **`xtrabackup`** all data is being compressed, including the transaction log file and meta data files, using the specified compression algorithm. The only currently supported algorithm is `quicklz`.

The resulting files have the qpress archive format, i.e., every `*.qp` file produced by **`xtrabackup`** is essentially a one-file qpress archive and can be extracted and uncompressed by the qpress file archiver. This means that there is no need to decompress entire backup to restore a single table as with `tar.gz`.

Files can be decompressed using the **qpress** tool that can be downloaded from here. Qpress supports multi-threaded decompression.

The xbcrypt binary

To support encryption and decryption of the backups, a new tool `xbcrypt` was introduced to *Percona XtraBackup*.

This utility has been modeled after *The xbstream binary* to perform encryption and decryption outside of *Percona XtraBackup*. `xbcrypt` has following command line options:

`-d, --decrypt`
Decrypt data input to output.

`-i, --input``=name`
Optional input file. If not specified, input will be read from standard input.

`-o, --output``=name`
Optional output file. If not specified, output will be written to standard output.

`-a, --encrypt-algo``=name`
Encryption algorithm.

`-k, --encrypt-key``=name`
Encryption key.

-f, --encrypt-key-file=name
File which contains encryption key.

-s, --encrypt-chunk-size=#
Size of working buffer for encryption in bytes. The default value is 64K.

--encrypt-threads=#
This option specifies the number of worker threads that will be used for parallel encryption/decryption.

-v, --verbose
Display verbose status output.

The xbcloud Binary

The purpose of *xbcloud* is to download and upload full or part of *xbstream* archive from/to cloud. *xbcloud* will not overwrite the backup with the same name. *xbcloud* accepts input via a pipe from *xbstream* so that it can be invoked as a pipeline with **xtrabackup** to stream directly to the cloud without needing a local storage.

xbcloud stores each chunk as a separate object with a name `backup_name/database/table.ibd.NNNNNNNNNNNNNNNNNNNN`, where `NNN...` is a 0-padded serial number of chunk within file. Size of chunk produced by **xtrabackup** and *xbstream* changed to 10M.

xbcloud has three essential operations: *put*, *get*, and *delete*. With these operations, backups are created, stored, retrieved, restored, and deleted. *xbcloud* operations clearly map to similar operations within the AWS S3 API.

Version specific information

- 2.4.14 - Added the support of Amazon S3, MinIO and Google Cloud Storage storage types.
- 2.3.1-beta1 - Implemented ability to store *xbcloud* parameters in a `.cnf` file
- 2.3.1-beta1 - Implemented support different *authentication options* for Swift
- 2.3.1-beta1 - Implemented support for partial download of the cloud backups
- 2.3.1-beta1 - *xbcloud --swift-url* option has been renamed to *xbcloud --swift-auth-url*
- 2.3.0-alpha1 - Initial implementation

Supported Cloud Storage Types

In addition to Swift, which has been the only option for storing backups in a cloud storage until *Percona XtraBackup* 2.4.14, *xbcloud* supports Amazon S3, MinIO, and Google Cloud Storage. Other Amazon S3 compatible storages, such as Wasabi or Digital Ocean Spaces, are also supported.

See also:

OpenStack Object Storage ("Swift") https://wiki.openstack.org/wiki/Swift

Amazon Simple Storage Service https://aws.amazon.com/s3/

MinIO https://min.io/

Google Cloud Storage https://cloud.google.com/storage/

Wasabi https://wasabi.com/

Digital Ocean Spaces https://www.digitalocean.com/products/spaces/

Usage

```
$ xtrabackup --backup --stream=xbstream --target-dir=/tmp | xbcloud \
put [options] <name>
```

Creating a full backup with Swift

The following example shows how to make a full backup and upload it to Swift.

```
$ xtrabackup --backup --stream=xbstream --extra-lsndir=/tmp --target-dir=/tmp | \
xbcloud put --storage=swift \
--swift-container=test \
--swift-user=test:tester \
--swift-auth-url=http://192.168.8.80:8080/ \
--swift-key=testing \
--parallel=10 \
full_backup
```

Creating a full backup with Amazon S3

```
$ xtrabackup --backup --stream=xbstream --extra-lsndir=/tmp --target-dir=/tmp | \
xbcloud put --storage=s3 \
--s3-endpoint='s3.amazonaws.com' \
--s3-access-key='YOUR-ACCESSKEYID' \
--s3-secret-key='YOUR-SECRETACCESSKEY' \
--s3-bucket='mysql_backups'
--parallel=10 \
${date -I}-full_backup
```

The following options are available when using Amazon S3:

Option	Details
–s3-access-key	Use to supply the AWS access key ID
–s3-secret-key	Use to supply the AWS secret access key
–s3-bucket	Use supply the AWS bucket name
–s3-region	Use to specify the AWS region. The default value is **us-east-1**
–s3-api-version = <AUTO|2|4>	Select the signing algorithm. The default value is AUTO. In this case, *xbcloud* will probe.
–s3-bucket-lookup = <AUTO|PATH|DNS>	Specify whether to use **bucket.endpoint.com** or *endpoint.com/bucket** style requests. The default value is AUTO. In this case, *xbcloud* will probe.

Creating a full backup with MinIO

```
$ xtrabackup --backup --stream=xbstream --extra-lsndir=/tmp --target-dir=/tmp | \
xbcloud put --storage=s3 \
--s3-endpoint='play.minio.io:9000' \
--s3-access-key='YOUR-ACCESSKEYID' \
--s3-secret-key='YOUR-SECRETACCESSKEY' \
--s3-bucket='mysql_backups'
--parallel=10 \
${date -I}-full_backup
```

Creating a full backup with Google Cloud Storage

The support for Google Cloud Storage is implemented using the interoperability mode. This mode was especially designed to interact with cloud services compatible with Amazon S3.

See also:

Cloud Storage Interoperability https://cloud.google.com/storage/docs/interoperability

```
$ xtrabackup --backup --stream=xbstream --extra-lsndir=/tmp --target-dir=/tmp | \
xbcloud put --storage=google \
--google-endpoint=`storage.googleapis.com` \
--google-access-key='YOUR-ACCESSKEYID' \
--google-secret-key='YOUR-SECRETACCESSKEY' \
--google-bucket='mysql_backups'
--parallel=10 \
${date -I}-full_backup
```

The following options are available when using Google Cloud Storage:

- –google-access-key = <ACCESS KEY ID>
- –google-secret-key = <SECRET ACCESS KEY>
- –google-bucket = <BUCKET NAME>

Supplying parameters

Each storage type has mandatory parameters that you can supply on the command line, in a configuration file, and via environment variables.

Configuration files

The parameters the values of which do not change frequently can be stored in `my.cnf` or in a custom configuration file. The following example is a template of configuration options under the `[xbcloud]` group:

```
[xbcloud]
storage=s3
s3-endpoint=http://localhost:9000/
s3-access-key=minio
s3-secret-key=minio123
s3-bucket=backupsx
s3-bucket-lookup=path
s3-api-version=4
```

Note: If you explicitly use a parameter on the command line and in a configuration file, *xbcloud* uses the the value provided on the command line.

Environment variables

The following environment variables are recognized. *xbcloud* maps them automatically to corresponding parameters applicable to the selected storage.

- AWS_ACCESS_KEY_ID (or ACCESS_KEY_ID)

- AWS_SECRET_ACCESS_KEY (or SECRET_ACCESS_KEY)
- AWS_DEFAULT_REGION (or DEFAULT_REGION)
- AWS_ENDPOINT (or ENDPOINT)
- AWS_CA_BUNDLE

Note: If you explicitly use a parameter on the command line, in a configuration file, and the corresponding environment variable contains a value, *xbcloud* uses the the value provided on the command line or in the configuration file.

OpenStack environment variables are also recognized and mapped automatically to corresponding **swift** parameters (`--storage=swift`).

- OS_AUTH_URL
- OS_TENANT_NAME
- OS_TENANT_ID
- OS_USERNAME
- OS_PASSWORD
- OS_USER_DOMAIN
- OS_USER_DOMAIN_ID
- OS_PROJECT_DOMAIN
- OS_PROJECT_DOMAIN_ID
- OS_REGION_NAME
- OS_STORAGE_URL
- OS_CACERT

Shortcuts

For all operations (put, get, and delete), you can use a shortcut to specify the storage type, bucket name, and backup name as one parameter instead of using three distinct parameters (–storage, –s3-bucket, and backup name per se).

Using a shortcut syntax to provide a storage type, bucket, and backup name

Use the following format: `storage-type://bucket-name/backup-name`

```
$ xbcloud get s3://operator-testing/bak22 ...
```

In this example, **s3** refers to a storage type, **operator-testing** is a bucket name, and **bak22** is the backup name. This shortcut expands as follows:

```
$ xbcloud get --storage=s3 --s3-bucket=operator-testing bak22 ...
```

You can supply the mandatory parameters not only on the command line. You may use configuration files and environment variables.

Additional parameters

xbcloud accepts additional parameters that you can use with any storage type. The `--md5` parameter computes the MD5 hash value of the backup chunks. The result is stored in files that following the `backup_name.md5` pattern.

```
$ xtrabackup --backup --stream=xbstream \
--parallel=8 2>backup.log | xbcloud put s3://operator-testing/bak22 \
--parallel=8 --md5 2>upload.log
```

You may use the `--header` parameter to pass an additional HTTP header with the server side encryption while specifying a customer key.

Example of using –header for AES256 encryption

```
$ xtrabackup --backup --stream=xbstream --parallel=4 | \
xbcloud put s3://operator-testing/bak-enc/ \
--header="X-Amz-Server-Side-Encryption-Customer-Algorithm: AES256" \
--header="X-Amz-Server-Side-Encryption-Customer-Key: CuStoMerKey=" \
--header="X-Amz-Server-Side-Encryption-Customer-Key-MD5: CuStoMerKeyMd5==" \
--parallel=8
```

The `--header` parameter is also useful to set the access control list (ACL) permissions: `--header="x-amz-acl: bucket-owner-full-control`

Restoring with Swift

```
$ xbcloud get [options] <name> [<list-of-files>] | xbstream -x
```

The following example shows how to fetch and restore the backup from Swift:

```
$ xbcloud get --storage=swift \
--swift-container=test \
--swift-user=test:tester \
--swift-auth-url=http://192.168.8.80:8080/ \
--swift-key=testing \
full_backup | xbstream -xv -C /tmp/downloaded_full

$ xtrabackup --prepare --target-dir=/tmp/downloaded_full
$ xtrabackup --copy-back --target-dir=/tmp/downloaded_full
```

Restoring with Amazon S3

```
$ xbcloud get s3://operator-testing/bak22 \
--s3-endpoint=https://storage.googleapis.com/ \
--parallel=10 2>download.log | xbstream -x -C restore --parallel=8
```

Incremental backups

First, you need to make the full backup on which the incremental one is going to be based:

```
$ xtrabackup --backup --stream=xbstream --extra-lsndir=/storage/backups/ \
--target-dir=/storage/backups/ | xbcloud put \
--storage=swift --swift-container=test_backup \
--swift-auth-version=2.0 --swift-user=admin \
--swift-tenant=admin --swift-password=xoxoxoxo \
```

```
--swift-auth-url=http://127.0.0.1:35357/ --parallel=10 \
full_backup
```

Then you can make the incremental backup:

```
$ xtrabackup --backup --incremental-basedir=/storage/backups \
--stream=xbstream --target-dir=/storage/inc_backup | xbcloud put \
--storage=swift --swift-container=test_backup \
--swift-auth-version=2.0 --swift-user=admin \
--swift-tenant=admin --swift-password=xoxoxoxo \
--swift-auth-url=http://127.0.0.1:35357/ --parallel=10 \
inc_backup
```

Preparing incremental backups

To prepare a backup you first need to download the full backup:

```
$ xbcloud get --swift-container=test_backup \
--swift-auth-version=2.0 --swift-user=admin \
--swift-tenant=admin --swift-password=xoxoxoxo \
--swift-auth-url=http://127.0.0.1:35357/ --parallel=10 \
full_backup | xbstream -xv -C /storage/downloaded_full
```

Once you download the full backup it should be prepared:

```
$ xtrabackup --prepare --apply-log-only --target-dir=/storage/downloaded_full
```

After the full backup has been prepared you can download the incremental backup:

```
$ xbcloud get --swift-container=test_backup \
--swift-auth-version=2.0 --swift-user=admin \
--swift-tenant=admin --swift-password=xoxoxoxo \
--swift-auth-url=http://127.0.0.1:35357/ --parallel=10 \
inc_backup | xbstream -xv -C /storage/downloaded_inc
```

Once the incremental backup has been downloaded you can prepare it by running:

```
$ xtrabackup --prepare --apply-log-only \
--target-dir=/storage/downloaded_full \
--incremental-dir=/storage/downloaded_inc

$ xtrabackup --prepare --target-dir=/storage/downloaded_full
```

Partial download of the cloud backup

If you don't want to download the entire backup to restore the specific database you can specify only tables you want to restore:

```
$ xbcloud get --swift-container=test_backup
--swift-auth-version=2.0 --swift-user=admin \
--swift-tenant=admin --swift-password=xoxoxoxo \
--swift-auth-url=http://127.0.0.1:35357/ full_backup \
ibdata1 sakila/payment.ibd \
> /storage/partial/partial.xbs
```

```
$ xbstream -xv -C /storage/partial < /storage/partial/partial.xbs
```

This command will download just `ibdata1` and `sakila/payment.ibd` table from the full backup.

Command-line options

xbcloud has the following command line options:

--storage=[swift|s3|google]
Cloud storage option. *xbcloud* supports Swift, MinIO, and AWS S3. The default value is `swift`.

--swift-auth-url
URL of Swift cluster.

--swift-url
Renamed to *xbcloud --swift-auth-url*

--swift-storage-url
xbcloud will try to get object-store URL for given region (if any specified) from the keystone response. One can override that URL by passing –swift-storage-url=URL argument.

--swift-user
Swift username (X-Auth-User, specific to Swift)

--swift-key
Swift key/password (X-Auth-Key, specific to Swift)

--swift-container
Container to backup into (specific to Swift)

--parallel=N
Maximum number of concurrent upload/download threads. Default is `1`.

--cacert
Path to the file with CA certificates

--insecure
Do not verify servers certificate

Swift authentication options

Swift specification describe several authentication options. *xbcloud* can authenticate against keystone with API version 2 and 3.

--swift-auth-version
Specifies the swift authentication version. Possible values are: `1.0` - TempAuth, `2.0` - Keystone v2.0, and `3` - Keystone v3. Default value is `1.0`.

For v2 additional options are:

--swift-tenant
Swift tenant name.

--swift-tenant-id
Swift tenant ID.

--swift-region
Swift endpoint region.

--swift-password
Swift password for the user.

For v3 additional options are:

--swift-user-id
Swift user ID.

--swift-project
Swift project name.

--swift-project-id
Swift project ID.

--swift-domain
Swift domain name.

--swift-domain-id
Swift domain ID.

Percona XtraBackup is a set of following tools:

innobackupex **innobackupex** is the symlink for **xtrabackup**. **innobackupex** still supports all features and syntax as 2.2 version did, but is now deprecated and will be removed in next major release.

xtrabackup a compiled *C* binary that provides functionality to backup a whole *MySQL* database instance with *MyISAM*, *InnoDB*, and *XtraDB* tables.

xbcrypt utility used for encrypting and decrypting backup files.

xbstream utility that allows streaming and extracting files to/from the *xbstream* format.

xbcloud utility used for downloading and uploading full or part of *xbstream* archive from/to cloud.

After *Percona XtraBackup* 2.3 release the recommend way to take the backup is using the **xtrabackup** script. More information on script options can be found in *how to use xtrabackup*.

Part VI

Advanced Features

CHAPTER

ELEVEN

LOCKLESS BINARY LOG INFORMATION

When the Lockless binary log information feature is available[1] on the server, *Percona XtraBackup* can trust binary log information stored in the *InnoDB* system header and avoid executing `LOCK BINLOG FOR BACKUP` (and thus, blocking commits for the duration of finalizing the `REDO` log copy) under a number of circumstances:

- when the server is not a GTID-enabled Galera cluster node
- when the replication I/O thread information should not be stored as a part of the backup (i.e. when the *xtrabackup --slave-info* option is not specified)

If all of the above conditions hold, *Percona XtraBackup* does not execute the `SHOW MASTER STATUS` as a part of the backup procedure, does not create the `xtrabackup_binlog_info` file on backup. Instead, that information is retrieved and the file is created after preparing the backup, along with creating `xtrabackup_binlog_pos_innodb`, which in this case contains exactly the same information as in `xtrabackup_binlog_info` and is thus redundant.

To make this new functionality configurable, there is now a new *Percona XtraBackup* option, *xtrabackup --binlog-info*, which can accept the following values:

- `OFF` - This means that *Percona XtraBackup* will not attempt to retrieve the binary log information at all, neither during the backup creation, nor after preparing it. This can help when a user just wants to copy data without any meta information like binary log or replication coordinates. In this case, `xtrabackup --binlog-info=OFF` can be passed to *Percona XtraBackup* and `LOCK BINLOG FOR BACKUP` will not be executed, even if the backup-safe binlog info feature is not provided by the server (but the backup locks feature is still a requirement).
- `ON` - This matches the old behavior, i.e. the one before this *Percona XtraBackup* feature had been implemented. When specified, *Percona XtraBackup* retrieves the binary log information and uses `LOCK BINLOG FOR BACKUP` (if available) to ensure its consistency.
- `LOCKLESS` - This corresponds to the functionality explained above: *Percona XtraBackup* will not retrieve binary log information during the backup process, will not execute `LOCK BINLOG FOR BACKUP`, and the `xtrabackup_binlog_info` file will not be created. The file will be created after preparing the backup using the information stored in the InnoDB system header. If the required server-side functionality is not provided by the server, specifying this *xtrabackup --binlog-info* value will result in an error. If one of the above mentioned conditions does not hold, `LOCK BINLOG FOR BACKUP` will still be executed to ensure consistency of other meta data.
- `AUTO` - This is the default value. When used, *Percona XtraBackup* will automatically switch to either `ON` or `LOCKLESS`, depending on the server-side feature availability, i.e., whether the `have_backup_safe_binlog_info` server variable is available.

[1] This feature is exclusive to *Percona Server for MySQL* starting with version 5.6.26-74.0. It is also used in *Percona XtraDB Cluster* when the node is being backed up without *xtrabackup --galera-info*.

CHAPTER

TWELVE

ENCRYPTED INNODB TABLESPACE BACKUPS

As of *MySQL* 5.7.11, InnoDB supports data encryption for InnoDB tables stored in file-per-table tablespaces. This feature provides at-rest encryption for physical tablespace data files.

For authenticated user or application to access encrypted tablespace, InnoDB will use master encryption key to decrypt the tablespace key. The master encryption key is stored in a keyring. Two keyring plugins supported by **xtrabackup** are `keyring_file` and `keyring_vault`. These plugins are installed into the plugin directory.

Making a Backup Using `keyring_file` Plugin

Support for encrypted InnoDB tablespace backups with `keyring_file` has been implemented in *Percona XtraBackup* 2.4.2 by implementing *`xtrabackup --keyring-file-data`* option (and also *`xtrabackup`*

--server-id option, needed for *MySQL* prior to 5.7.13). These options are only recognized by **xtrabackup** binary i.e., **innobackupex** will not be able to backup and prepare encrypted tablespaces.

Creating Backup

In order to backup and prepare database containing encrypted InnoDB tablespaces, you must specify the path to keyring file by using the *xtrabackup --keyring-file-data* option.

```
$ xtrabackup --backup --target-dir=/data/backup/ --user=root \
--keyring-file-data=/var/lib/mysql-keyring/keyring
```

With *MySQL* prior to 5.7.13, use *xtrabackup --server-id* in the backup creation command:

```
$ xtrabackup --backup --target-dir=/data/backup/ --user=root \
--keyring-file-data=/var/lib/mysql-keyring/keyring --server-id=1
```

After **xtrabackup** is finished taking the backup you should see the following message:

```
xtrabackup: Transaction log of lsn (5696709) to (5696718) was copied.
160401 10:25:51 completed OK!
```

Warning: **xtrabackup** will not copy keyring file into the backup directory. In order to be prepare the backup, you must make a copy of keyring file yourself.

Preparing Backup

In order to prepare the backup you'll need to specify the keyring-file-data (server-id is stored in `backup-my.cnf` file, so it can be omitted when preparing the backup, regardless of the *MySQL* version used).

```
xtrabackup --prepare --target-dir=/data/backup \
--keyring-file-data=/var/lib/mysql-keyring/keyring
```

After **xtrabackup** is finished preparing the backup you should see the following message:

```
InnoDB: Shutdown completed; log sequence number 5697064
160401 10:34:28 completed OK!
```

Backup is now prepared and can be restored with *xtrabackup --copy-back* option. In case the keyring has been rotated you'll need to restore the keyring which was used to take and prepare the backup.

Making Backup Using `keyring_vault` Plugin

Support for encrypted InnoDB tablespace backups with `keyring_vault` has been implemented in *Percona XtraBackup* 2.4.11. Keyring vault plugin settings are described here.

Creating Backup

The following command creates a backup in the `/data/backup` directory:

```
$ xtrabackup --backup --target-dir=/data/backup --user=root
```

After **xtrabackup** completes taking the backup you should see the following message:

```
xtrabackup: Transaction log of lsn (5696709) to (5696718) was copied.
160401 10:25:51 completed OK!
```

Preparing the Backup

In order to prepare the backup **xtrabackup** will need an access to the keyring. Since **xtrabackup** doesn't talk to MySQL server and doesn't read default `my.cnf` configuration file during prepare, user will need to specify keyring settings via the command line:

```
$ xtrabackup --prepare --target-dir=/data/backup \
--keyring-vault-config=/etc/vault.cnf
```

See also:

Data at Rest Encryption for Percona Server keyring vault plugin settings

After **xtrabackup** completes preparing the backup you should see the following message:

```
InnoDB: Shutdown completed; log sequence number 5697064
160401 10:34:28 completed OK!
```

The backup is now prepared and can be restored with *`xtrabackup --copy-back`* option:

```
xtrabackup --copy-back --target-dir=/data/backup --datadir=/data/mysql
```

Incremental Encrypted InnoDB Tablespace Backups with `keyring_file`

The process of taking incremental backups with InnoDB tablespace encryption is similar to taking the *Incremental Backups* with unencrypted tablespace.

Creating an Incremental Backup

To make an incremental backup, begin with a full backup. The **xtrabackup** binary writes a file called `xtrabackup_checkpoints` into the backup's target directory. This file contains a line showing the `to_lsn`, which is the database's *LSN* at the end of the backup. First you need to create a full backup with the following command:

```
$ xtrabackup --backup --target-dir=/data/backups/base \
--keyring-file-data=/var/lib/mysql-keyring/keyring
```

Warning: **xtrabackup** will not copy keyring file into the backup directory. In order to be prepare the backup, you must make a copy of keyring file yourself. If you try to restore the backup after the keyring has been changed you'll see errors like `ERROR 3185 (HY000): Can't find master key from keyring, please check keyring plugin is loaded.` when trying to access encrypted table.

If you look at the `xtrabackup_checkpoints` file, you should see some contents similar to the following:

```
backup_type = full-backuped
from_lsn = 0
to_lsn = 7666625
last_lsn = 7666634
compact = 0
recover_binlog_info = 1
```

Now that you have a full backup, you can make an incremental backup based on it. Use a command such as the following:

```
$ xtrabackup --backup --target-dir=/data/backups/inc1 \
--incremental-basedir=/data/backups/base \
--keyring-file-data=/var/lib/mysql-keyring/keyring
```

Warning: **`xtrabackup`** will not copy keyring file into the backup directory. In order to be prepare the backup, you must make a copy of keyring file yourself. If the keyring hasn't been rotated you can use the same as the one you've backed-up with the base backup. If the keyring has been rotated you'll need to back it up otherwise you won't be able to prepare the backup.

The `/data/backups/inc1/` directory should now contain delta files, such as `ibdata1.delta` and `test/table1.ibd.delta`. These represent the changes since the `LSN 7666625`. If you examine the `xtrabackup_checkpoints` file in this directory, you should see something similar to the following:

```
backup_type = incremental
from_lsn = 7666625
to_lsn = 8873920
last_lsn = 8873929
compact = 0
recover_binlog_info = 1
```

The meaning should be self-evident. It's now possible to use this directory as the base for yet another incremental backup:

```
$ xtrabackup --backup --target-dir=/data/backups/inc2 \
--incremental-basedir=/data/backups/inc1 \
--keyring-file-data=/var/lib/mysql-keyring/keyring
```

Preparing the Incremental Backups

The *`xtrabackup --prepare`* step for incremental backups is not the same as for normal backups. In normal backups, two types of operations are performed to make the database consistent: committed transactions are replayed from the log file against the data files, and uncommitted transactions are rolled back. You must skip the rollback of uncommitted transactions when preparing a backup, because transactions that were uncommitted at the time of your backup may be in progress, and it's likely that they will be committed in the next incremental backup. You should use the *`xtrabackup --apply-log-only`* option to prevent the rollback phase.

Warning: If you do not use the *`xtrabackup --apply-log-only`* option to prevent the rollback phase, then your incremental backups will be useless. After transactions have been rolled back, further incremental backups cannot be applied.

Beginning with the full backup you created, you can prepare it, and then apply the incremental differences to it. Recall that you have the following backups:

```
/data/backups/base
/data/backups/inc1
/data/backups/inc2
```

To prepare the base backup, you need to run *xtrabackup --prepare* as usual, but prevent the rollback phase:

```
$ xtrabackup --prepare --apply-log-only --target-dir=/data/backups/base \
--keyring-file-data=/var/lib/mysql-keyring/keyring
```

The output should end with some text such as the following:

```
InnoDB: Shutdown completed; log sequence number 7666643
InnoDB: Number of pools: 1
160401 12:31:11 completed OK!
```

To apply the first incremental backup to the full backup, you should use the following command:

```
$ xtrabackup --prepare --apply-log-only --target-dir=/data/backups/base \
--incremental-dir=/data/backups/inc1 \
--keyring-file-data=/var/lib/mysql-keyring/keyring
```

Warning: Backup should be prepared with the keyring that was used when backup was being taken. This means that if the keyring has been rotated between the base and incremental backup that you'll need to use the keyring that was in use when the first incremental backup has been taken.

Preparing the second incremental backup is a similar process: apply the deltas to the (modified) base backup, and you will roll its data forward in time to the point of the second incremental backup:

```
$ xtrabackup --prepare --target-dir=/data/backups/base \
--incremental-dir=/data/backups/inc2 \
--keyring-file-data=/var/lib/mysql-keyring/keyring
```

Note: *xtrabackup --apply-log-only* should be used when merging all incrementals except the last one. That's why the previous line doesn't contain *xtrabackup --apply-log-only*. Even if the *xtrabackup --apply-log-only* were used on the last step, backup would still be consistent but in that case server would perform the rollback phase.

The backup is now prepared and can be restored with *xtrabackup --copy-back*. In case the keyring has been rotated you'll need to restore the keyring which was used to take and prepare the backup.

Restoring backup when keyring is not available

While described restore method works, it requires an access to the same keyring which server is using. It may not be possible if backup is prepared on different server or at the much later time, when keys in the keyring have been purged, or in case of malfunction when keyring vault server is not available at all.

A *xtrabackup --transition-key* should be used to make it possible for **xtrabackup** to process the backup without access to the keyring vault server. In this case **xtrabackup** will derive AES encryption key from specified passphrase and will use it to encrypt tablespace keys of tablespaces being backed up.

Creating the Backup with a Passphrase

The following example illustrates how a backup can be created in this case:

```
$ xtrabackup --backup --user=root -p --target-dir=/data/backup \
--transition-key=MySecetKey
```

If *xtrabackup --transition-key* is specified without a value, xtrabackup will ask for it.

Note: *xtrabackup --transition-key* scrapes the supplkied value so that it should not visible in the `ps` command output.

Preparing the Backup with a Passphrase

The same passphrase should be specified for the `prepare` command:

```
$ xtrabackup --prepare --target-dir=/data/backup
```

There is no `keyring-vault` or `keyring-file` here, because **xtrabackup** does not talk to the keyring in this case.

Restoring the Backup with a Generated Key

When restoring a backup you will need to generate new master key. Here is the example for `keyring_file`:

```
$ xtrabackup --copy-back --target-dir=/data/backup --datadir=/data/mysql \
--transition-key=MySecetKey --generate-new-master-key \
--keyring-file-data=/var/lib/mysql-keyring/keyring
```

In case of `keyring_vault` it will look like this:

```
$ xtrabackup --copy-back --target-dir=/data/backup --datadir=/data/mysql \
--transition-key=MySecetKey --generate-new-master-key \
--keyring-vault-config=/etc/vault.cnf
```

xtrabackup will generate new master key, store it into target keyring vault server and re-encrypt tablespace keys using this key.

Making the Backup with a Stored Transition Key

Finally, there is an option to store transition key in the keyring. In this case **xtrabackup** will need an access to the same keyring file or vault server during prepare and copy-back, but does not depend on whether the server keys have been purged.

In this scenario, the three stages of the backup process are the following:

- *Backup*
- *Prepare*

Backup

```
$ xtrabackup --backup --user=root -p --target-dir=/data/backup \
--generate-transition-key
```

Prepare

`keyring_file` variant

```
$ xtrabackup --prepare --target-dir=/data/backup \
--keyring-file-data=/var/lib/mysql-keyring/keyring
```

`keyring_vault` variant

```
$ xtrabackup --prepare --target-dir=/data/backup \
--keyring-vault-config=/etc/vault.cnf
```

Copy-back

`keyring_file` variant

```
$ xtrabackup --copy-back --target-dir=/data/backup --datadir=/data/mysql \
--generate-new-master-key --keyring-file-data=/var/lib/mysql-keyring/keyring
```

`keyring_vault` variant

```
$ xtrabackup --copy-back --target-dir=/data/backup --datadir=/data/mysql \
--generate-new-master-key --keyring-vault-config=/etc/vault.cnf
```

Part VII

Tutorials, Recipes, How-tos

CHAPTER

THIRTEEN

HOW-TOS AND RECIPES

Recipes for `innobackupex`

Make a Local Full Backup (Create, Prepare and Restore)

Create the Backup

This is the simplest use case. It copies all your *MySQL* data into the specified directory. Here is how to make a backup of all the databases in the *datadir* specified in your *my.cnf*. It will put the backup in a time stamped subdirectory of `/data/backups/`, in this case, `/data/backups/2010-03-13_02-42-44`,

```
$ innobackupex /data/backups
```

There is a lot of output, but you need to make sure you see this at the end of the backup. If you don't see this output, then your backup failed:

```
100313 02:43:07  innobackupex: completed OK!
```

Prepare the Backup

To prepare the backup use the *innobackupex --apply-log* option and specify the timestamped subdirectory of the backup. To speed up the apply-log process, use the *innobackupex --use-memory*:

```
$ innobackupex --use-memory=4G --apply-log /data/backups/2010-03-13_02-42-44/
```

You should check for a confirmation message:

```
100313 02:51:02  innobackupex: completed OK!
```

Now the files in `/data/backups/2010-03-13_02-42-44` is ready to be used by the server.

Restore the Backup

To restore the already-prepared backup, first stop the server and then use the *innobackupex --copy-back* function of **innobackupex**:

```
innobackupex --copy-back /data/backups/2010-03-13_02-42-44/
## Use chmod to correct the permissions, if necessary!
```

This will copy the prepared data back to its original location as defined by the `datadir` in your *my.cnf*.

Note: The *datadir* must be empty; *Percona XtraBackup* `innobackupex --copy-back` option will not copy over existing files unless `innobackupex --force-non-empty-directories` option is specified. Also it's important to note that *MySQL* server needs to be shut down before restore is performed. You can't restore to a *datadir* of a running mysqld instance (except when importing a partial backup).

After the confirmation message:

```
100313 02:58:44  innobackupex: completed OK!
```

you should check the file permissions after copying the data back. You may need to adjust them with something like:

```
$ chown -R mysql:mysql /var/lib/mysql
```

Now the *datadir* contains the restored data. You are ready to start the server.

Make a Streaming Backup

Stream mode sends the backup to `STDOUT` in the xbstream format instead of copying it to the directory named by the first argument. You can pipe the output to **`gzip`**, or across the network, to another server.

To extract the resulting xbstream file, you **must** use the *xbstream* utility: `xbstream -x < backup.xbstream`.

Here are some examples of using the `xbstream` option for streaming:

Action	Command
Stream the backup into a xbstream archived named 'backup.xbstream'	$ xtrabackup –backup –stream=xbstream ./ > backup.xbstream
Stream the backup into a xbstream archived named 'backup.xbstream' and compress it	$ xtrabackup –backup –stream=xbstream –compress ./ > backup.xbstream
Encrypt the backup	$ xtrabackup –backup –stream=xbstream ./ > backup.xbstream gzip - | openssl des3 -salt -k "password" > backup.xbstream.gz.des3
Send the backup to another server and unpack it	$ xtrabackup –backup –compress –stream=xbstream ./ | ssh user@otherhost "xbstream -x"

Send the backup to another server using `netcat`.	On the destination host:
	`$ nc -l 9999 | cat - > /data/backups/backup.xbstream`
	On the source host:
	`$ xtrabackup --backup --stream=xbstream ./ | nc desthost 9999`

Send the backup to another server using a one-liner:	$ ssh user@desthost "(nc -l 9999 > /data/backups/backup.xbstream &)" && xtrabackup –backup –stream=xbstream ./ | nc desthost 9999
Throttle the throughput to 10MB/sec using the `pipe viewer` tool[1]	$ xtrabackup –backup –stream=xbstream ./ | pv -q -L10m ssh user@desthost "cat - > /data/backups/backup.xbstream"

Checksumming the backup during the streaming:	On the destination host:
	`$ nc -l 9999 | tee >(sha1sum > destination_checksum) > /data/backups/backup.xbstream`
	On the source host:
	`$ xtrabackup --backup --stream=xbstream ./ | tee >(sha1sum > source_checksum) | nc desthost 9999`
	Compare the checksums on the source host:
	`$ cat source_checksum`
	`65e4f916a49c1f216e0887ce54cf59bf3934dbad -`
	Compare the checksums on the destination host:
	`$ cat destination_checksum`
	`65e4f916a49c1f216e0887ce54cf59bf3934dbad -`

Parallel compression with parallel copying backup:	$ xtrabackup –backup –compress –compress-threads=8 –stream=xbstream –parallel=4 ./ > backup.xbstream

Making an Incremental Backup

Every incremental backup starts with a full one, which we will call the *base backup*:

```
innobackupex --user=USER --password=PASSWORD /path/to/backup/dir/
```

Note that the full backup will be in a timestamped subdirectory of `/path/to/backup/dir/` (e.g. `/path/to/backup/dir/2011-12-24_23-01-00/`).

[1] Install from the official site or from the distribution package (`apt install pv`)

Assuming that variable `$FULLBACKUP` contains `/path/to/backup/dir/2011-5-23_23-01-18`, let's do an incremental backup an hour later:

```
innobackupex --incremental /path/to/inc/dir \
  --incremental-basedir=$FULLBACKUP --user=USER --password=PASSWORD
```

Now, the incremental backup should be in `/path/to/inc/dir/2011-12-25_00-01-00/`. Let's call `$INCREMENTALBACKUP=2011-5-23_23-50-10`.

Preparing incremental backups is a bit different than full ones:

First you have to replay the committed transactions on each backup,

```
innobackupex --apply-log --redo-only $FULLBACKUP \
 --use-memory=1G --user=USER --password=PASSWORD
```

The *innobackupex --use-memory* option is not necessary, it will speed up the process if it is used (provided that the amount of RAM given is available).

If everything went fine, you should see an output similar to:

```
111225 01:10:12 InnoDB: Shutdown completed; log sequence number 91514213
```

Now apply the incremental backup to the base backup, by issuing:

```
innobackupex --apply-log --redo-only $FULLBACKUP
 --incremental-dir=$INCREMENTALBACKUP
 --use-memory=1G --user=DVADER --password=D4RKS1D3
```

Note the `$INCREMENTALBACKUP`.

***The final data will be in the base backup directory*, not in the incremental** one. In this example, `/path/to/backup/dir/2011-12-24_23-01-00` or `$FULLBACKUP`.

If you want to apply more incremental backups, repeat this step with the next one. It is important that you do this in the chronological order in which the backups were done.

You can check the file xtrabackup_checkpoints at the directory of each one.

They should look like: (in the base backup)

```
backup_type = full-backuped
from_lsn = 0
to_lsn = 1291135
```

and in the incremental ones:

```
backup_type = incremental
from_lsn = 1291135
to_lsn = 1291340
```

The `to_lsn` number must match the `from_lsn` of the next one.

Once you put all the parts together, you can prepare again the full backup (base + incrementals) once again to rollback the pending transactions:

```
$ innobackupex-1.5.1 --apply-log $FULLBACKUP --use-memory=1G \
--user=$USERNAME --password=$PASSWORD
```

Now your backup is ready to be used immediately after restoring it. This preparation step is optional, as if you restore it without doing it, the database server will assume that a crash occurred and will begin to rollback the uncommitted transaction (causing some downtime which can be avoided).

Making a Compressed Backup

In order to make a compressed backup you'll need to use *innobackupex --compress*

```
$ innobackupex --compress /data/backup
```

If you want to speed up the compression you can use the parallel compression, which can be enabled with *innobackupex --compress-threads* option. Following example will use four threads for compression:

```
$ innobackupex --compress --compress-threads=4 /data/backup
```

Output should look like this

```
...
[01] Compressing ./imdb/comp_cast_type.ibd to /data/backup/2013-08-01_11-24-04/./imdb/
↪comp_cast_type.ibd.qp
[01]        ...done
[01] Compressing ./imdb/aka_name.ibd to /data/backup/2013-08-01_11-24-04/./imdb/aka_
↪name.ibd.qp
[01]        ...done
...
130801 11:50:24  innobackupex: completed OK
```

Preparing the backup

Before you can prepare the backup you'll need to uncompress all the files with qpress (which is available from Percona Software repositories). You can use following one-liner to uncompress all the files:

```
$ for bf in `find . -iname "*\.qp"`; do qpress -d $bf $(dirname $bf) && rm $bf; done
```

In *Percona XtraBackup* 2.1.4 new *innobackupex --decompress* option has been implemented that can be used to decompress the backup:

```
$ innobackupex --decompress /data/backup/2013-08-01_11-24-04/
```

Note: In order to successfully use the *innobackupex --decompress* option, qpress binary needs to installed and within the path. *innobackupex --parallel* can be used with *innobackupex --decompress* option to decompress multiple files simultaneously.

When the files are uncompressed you can prepare the backup with *innobackupex --apply-log*:

```
$ innobackupex --apply-log /data/backup/2013-08-01_11-24-04/
```

You should check for a confirmation message:

```
130802 02:51:02  innobackupex: completed OK!
```

Now the files in `/data/backups/2013-08-01_11-24-04` is ready to be used by the server.

Note: *Percona XtraBackup* doesn't automatically remove the compressed files. In order to clean up the backup directory users should remove the `*.qp` files.

Restoring the backup

Once the backup has been prepared you can use the *innobackupex --copy-back* to restore the backup.

```
$ innobackupex --copy-back /data/backups/2013-08-01_11-24-04/
```

This will copy the prepared data back to its original location as defined by the *datadir* in your *my.cnf*.

After the confirmation message:

```
130802 02:58:44  innobackupex: completed OK!
```

you should check the file permissions after copying the data back. You may need to adjust them with something like:

```
$ chown -R mysql:mysql /var/lib/mysql
```

Now the *datadir* contains the restored data. You are ready to start the server.

Backing Up and Restoring Individual Partitions

Percona XtraBackup features *partial backups*, which means that you may backup individual partitions as well because from the storage engines perspective partitions are regular tables with specially formatted names. The only requirement for this feature is having the *innodb_file_per_table* option enabled in the server.

There is only one caveat about using this kind of backup: you can't copy back the prepared backup. Restoring partial backups should be done by importing the tables, and not by using the traditional *innobackupex --copy-back* option. Although there are some scenarios where restoring can be done by copying back the files, this may be lead to database inconsistencies in many cases and it is not the recommended way to do it.

Creating the backup

There are three ways of specifying which part of the whole data will be backed up: regular expressions (*innobackupex --include*), enumerating the tables in a file (*innobackupex --tables-file*) or providing a list of databases (*innobackupex --databases*). In this example *innobackupex --include* option will be used.

The regular expression provided to this option will be matched against the fully qualified tablename, including the database name, in the form `databasename.tablename`.

For example, this will back up the partition `p4` from the table `name` located in the database `imdb`:

```
$ innobackupex --include='^imdb[.]name#P#p4' /mnt/backup/
```

This will create a timestamped directory with the usual files that **innobackupex** creates, but only the data files related to the tables matched.

Output of the **innobackupex** will list the skipped tables

```
...
[01] Skipping ./imdb/person_info.ibd
[01] Skipping ./imdb/name#P#p5.ibd
[01] Skipping ./imdb/name#P#p6.ibd
...
imdb.person_info.frm is skipped because it does not match ^imdb[.]name#P#p4.
imdb.title.frm is skipped because it does not match ^imdb[.]name#P#p4.
imdb.company_type.frm is skipped because it does not match ^imdb[.]name#P#p4.
...
```

Note that this option is passed to *xtrabackup --tables* and is matched against each table of each database, the directories of each database will be created even if they are empty.

Preparing the backup

For preparing partial backups, the procedure is analogous to *restoring individual tables* : apply the logs and use *innobackupex --export*:

```
$ innobackupex --apply-log --export /mnt/backup/2012-08-28_10-29-09
```

You may see warnings in the output about tables that don't exists. This is because *InnoDB*-based engines stores its data dictionary inside the tablespace files besides the *.frm* files. **innobackupex** will use **xtrabackup** to remove the missing tables (those that haven't been selected in the partial backup) from the data dictionary in order to avoid future warnings or errors:

```
InnoDB: in InnoDB data dictionary has tablespace id 51,
InnoDB: but tablespace with that id or name does not exist. It will be removed from␣
↪data dictionary.
120828 10:25:28  InnoDB: Waiting for the background threads to start
120828 10:25:29 Percona XtraDB (http://www.percona.com) 1.1.8-20.1 started; log␣
↪sequence number 10098323731
xtrabackup: export option is specified.
xtrabackup: export metadata of table 'imdb/name#P#p4' to file `./imdb/name#P#p4.exp`␣
↪(1 indexes)
xtrabackup:     name=PRIMARY, id.low=73, page=3
```

You should also see the notification of the creation of a file needed for importing (*.exp* file) for each table included in the partial backup:

```
xtrabackup: export option is specified.
xtrabackup: export metadata of table 'imdb/name#P#p4' to file `./imdb/name#P#p4.exp`␣
↪(1 indexes)
xtrabackup:     name=PRIMARY, id.low=73, page=3
```

Note that you can use *innobackupex --export* with *innobackupex --apply-log* to an already-prepared backup in order to create the *.exp* files.

Finally, check the for the confirmation message in the output:

```
120828 19:25:38  innobackupex: completed OK!
```

Restoring from the backups

Restoring should be done by *importing the tables* in the partial backup to the server.

Note: Improved table/partition import is only available in *Percona Server for MySQL* and *MySQL* 5.6, this means that partitions which were backed up from different server can be imported as well. For versions older than *MySQL* 5.6 only partitions from that server can be imported with some important limitations. There should be no DROP/CREATE/TRUNCATE/ALTER TABLE commands issued between taking the backup and importing the partition.

First step is to create new table in which data will be restored

```
.. code-block:: guess
```

mysql> CREATE TABLE *name_p4* (*id* int(11) NOT NULL AUTO_INCREMENT, *name* text NOT NULL, *imdb_index* varchar(12) DEFAULT NULL, *imdb_id* int(11) DEFAULT NULL, *name_pcode_cf* varchar(5) DEFAULT NULL, *name_pcode_nf* varchar(5) DEFAULT NULL, *surname_pcode* varchar(5) DEFAULT NULL, PRIMARY KEY (*id*)) ENGINE=InnoDB AUTO_INCREMENT=2812744 DEFAULT CHARSET=utf8

To restore the partition from the backup tablespace needs to be discarded for that table:

```
mysql>  ALTER TABLE name_p4 DISCARD TABLESPACE;
```

The next step is to copy the *.exp* and *ibd* files from the backup to *MySQL* data directory:

```
$ cp /mnt/backup/2012-08-28_10-29-09/imdb/name#P#p4.exp /var/lib/mysql/imdb/name_p4.
↪exp
$ cp /mnt/backup/2012-08-28_10-29-09/imdb/name#P#p4.ibd /var/lib/mysql/imdb/name_p4.
↪ibd
```

Note: Make sure that the copied files can be accessed by the user running the *MySQL*.

If you are running the *Percona Server for MySQL* make sure that variable *innodb_import_table_from_xtrabackup* is enabled:

```
mysql> SET GLOBAL innodb_import_table_from_xtrabackup=1;
```

The last step is to import the tablespace:

```
mysql>  ALTER TABLE name_p4 IMPORT TABLESPACE;
```

Restoring from the backups in version 5.6

The problem with server versions up to 5.5 is that there is no server support to import either individual partitions or all partitions of a partitioned table, so partitions could only be imported as independent tables. In *MySQL* and *Percona Server for MySQL* 5.6 it is possible to exchange individual partitions with independent tables through `ALTER TABLE ... EXCHANGE PARTITION` command.

Note: In *Percona Server for MySQL* 5.6, the variable `innodb_import_table_from_xtrabackup` was been removed in favor of *MySQL* Transportable Tablespaces implementation.

When importing an entire partitioned table, first import all (sub)partitions as independent tables:

```
mysql> CREATE TABLE `name_p4` (
`id` int(11) NOT NULL AUTO_INCREMENT,
`name` text NOT NULL,
`imdb_index` varchar(12) DEFAULT NULL,
`imdb_id` int(11) DEFAULT NULL,
`name_pcode_cf` varchar(5) DEFAULT NULL,
`name_pcode_nf` varchar(5) DEFAULT NULL,
`surname_pcode` varchar(5) DEFAULT NULL,
PRIMARY KEY (`id`)
) ENGINE=InnoDB AUTO_INCREMENT=2812744 DEFAULT CHARSET=utf8
```

To restore the partition from the backup tablespace needs to be discarded for that table:

```
mysql>  ALTER TABLE name_p4 DISCARD TABLESPACE;
```

The next step is to copy the `.cfg` and `.ibd` files from the backup to *MySQL* data directory:

```
$ cp /mnt/backup/2013-07-18_10-29-09/imdb/name#P#p4.cfg /var/lib/mysql/imdb/name_p4.
↪cfg
$ cp /mnt/backup/2013-07-18_10-29-09/imdb/name#P#p4.ibd /var/lib/mysql/imdb/name_p4.
↪ibd
```

The last step is to import the tablespace:

```
mysql>  ALTER TABLE name_p4 IMPORT TABLESPACE;
```

We can now create the empty partitioned table with exactly the same schema as the table being imported:

```
mysql> CREATE TABLE name2 LIKE name;
```

Then swap empty partitions from the newly created table with individual tables corresponding to partitions that have been exported/imported on the previous steps:

```
mysql> ALTER TABLE name2 EXCHANGE PARTITION p4 WITH TABLE name_p4;
```

In order for this operation to be successful following conditions have to be met.

Recipes for `xtrabackup`

Making a Full Backup

Backup the InnoDB data and log files - located in `/var/lib/mysql/` - to `/data/backups/mysql/` (destination). Then, prepare the backup files to be ready to restore or use (make the data files consistent).

Make a backup:

```
$ xtrabackup --backup --target-dir=/data/backups/mysql/
```

Prepare the backup twice:

```
$ xtrabackup --prepare --target-dir=/data/backups/mysql/
$ xtrabackup --prepare --target-dir=/data/backups/mysql/
```

Success Criterion

- The exit status of xtrabackup is 0.
- In the second *xtrabackup --prepare* step, you should see InnoDB print messages similar to `Log file ./ib_logfile0 did not exist: new to be created`, followed by a line indicating the log file was created (creating new logs is the purpose of the second preparation).

Notes

- You might want to set the *xtrabackup --use-memory* option to something similar to the size of your buffer pool, if you are on a dedicated server that has enough free memory. More details *here*.
- For a more detailed explanation, see *Creating a backup*

Making an Incremental Backup

Backup all the InnoDB data and log files - located in `/var/lib/mysql/` - **once**, then make two daily incremental backups in `/data/backups/mysql/` (destination). Finally, prepare the backup files to be ready to restore or use.

Create one full backup

Making an incremental backup requires a full backup as a base:

```
xtrabackup --backup --target-dir=/data/backups/mysql/
```

It is important that you **do not run** the *xtrabackup --prepare* command yet.

Create two incremental backups

Suppose the full backup is on Monday, and you will create an incremental one on Tuesday:

```
xtrabackup --backup --target-dir=/data/backups/inc/tue/ \
      --incremental-basedir=/data/backups/mysql/
```

and the same policy is applied on Wednesday:

```
xtrabackup --backup --target-dir=/data/backups/inc/wed/ \
       --incremental-basedir=/data/backups/inc/tue/
```

Prepare the base backup

Prepare the base backup (Monday's backup):

```
xtrabackup --prepare --apply-log-only --target-dir=/data/backups/mysql/
```

Roll forward the base data to the first increment

Roll Monday's data forward to the state on Tuesday:

```
xtrabackup --prepare --apply-log-only --target-dir=/data/backups/mysql/ \
   --incremental-dir=/data/backups/inc/tue/
```

Roll forward again to the second increment

Roll forward again to the state on Wednesday:

```
xtrabackup --prepare --apply-log-only --target-dir=/data/backups/mysql/ \
   --incremental-dir=/data/backups/inc/wed/
```

Prepare the whole backup to be ready to use

Create the new logs by preparing it:

```
xtrabackup --prepare --target-dir=/data/backups/mysql/
```

Notes

- You might want to set the *`xtrabackup --use-memory`* to speed up the process if you are on a dedicated server that has enough free memory. More details *here*.
- A more detailed explanation is *here*.

Restoring the Backup

Because **`xtrabackup`** doesn't copy *MyISAM* files, `.frm` files, and the rest of the database, you might need to back those up separately. To restore the InnoDB data, simply do something like the following after preparing:

```
cd /data/backups/mysql/
rsync -rvt --exclude 'xtrabackup_checkpoints' --exclude 'xtrabackup_logfile' \
   ./ /var/lib/mysql
chown -R mysql:mysql /var/lib/mysql/
```

How-Tos

How to setup a slave for replication in 6 simple steps with Percona XtraBackup

Data is, by far, the most valuable part of a system. Having a backup done systematically and available for a rapid recovery in case of failure is admittedly essential to a system. However, it is not common practice because of its costs, infrastructure needed or even the boredom associated to the task. *Percona XtraBackup* is designed to solve this problem.

You can have almost real-time backups in 6 simple steps by setting up a replication environment with *Percona XtraBackup*.

Percona XtraBackup is a tool for backing up your data extremely easy and without interruption. It performs "hot backups" on unmodified versions of *MySQL* servers (5.1, 5.5 and 5.6), as well as *MariaDB* and *Percona Servers*. It is a totally free and open source software distributed only under the *GPLv2* license.

All the things you will need

Setting up a slave for replication with *Percona XtraBackup* is really a very straightforward procedure. In order to keep it simple, here is a list of the things you need to follow the steps without hassles:

- `TheMaster` A system with a *MySQL*-based server installed, configured and running. This system will be called `TheMaster`, as it is where your data is stored and the one to be replicated. We will assume the following about this system:
  - the *MySQL* server is able to communicate with others by the standard TCP/IP port;
  - the *SSH* server is installed and configured;
  - you have a user account in the system with the appropriate permissions;
  - you have a MySQL's user account with appropriate privileges.
  - server has binlogs enabled and server-id set up to 1.
- `TheSlave` Another system, with a *MySQL*-based server installed on it. We will refer to this machine as `TheSlave` and we will assume the same things we did about `TheMaster`, except that the server-id on `TheSlave` is 2.
- `Percona XtraBackup` The backup tool we will use. It should be installed in both computers for convenience.

Note: It is not recommended to mix MySQL variants (Percona Server, MySQL, MariaDB) in your replication setup. This may produce incorrect `xtrabackup_slave_info` file when adding a new slave.

STEP 1: Make a backup on `TheMaster` and prepare it

At `TheMaster`, issue the following to a shell:

```
TheMaster$ xtrabackup --backup --user=yourDBuser --password=MaGiCdB1 --target-dir=/
↪path/to/backupdir
```

After this is finished you should get:

```
xtrabackup: completed OK!
```

This will make a copy of your *MySQL* data dir to the `/path/to/backupdir` directory. You have told *Percona XtraBackup* to connect to the database server using your database user and password, and do a hot backup of all your data in it (all *MyISAM*, *InnoDB* tables and indexes in them).

In order for snapshot to be consistent you need to prepare the data:

```
TheMaster$ xtrabackup --user=yourDBuser --password=MaGiCdB1 \
             --prepare --target-dir=/path/to/backupdir
```

You need to select path where your snapshot has been taken. If everything is ok you should get the same OK message. Now the transaction logs are applied to the data files, and new ones are created: your data files are ready to be used by the MySQL server.

Percona XtraBackup knows where your data is by reading your *my.cnf*. If you have your configuration file in a non-standard place, you should use the flag `xtrabackup --defaults-file=/location/of/my.cnf`.

If you want to skip writing the user name and password every time you want to access *MySQL*, you can set it up in `.mylogin.cnf` as follows:

```
mysql_config_editor set --login-path=client --host=localhost --user=root --password
```

This will give you the root access to MySQL.

See also:

***MySQL* Documentaiton: MySQL Configuration Utility** https://dev.mysql.com/doc/refman/5.7/en/mysql-config-editor.html

STEP 2: Copy backed up data to TheSlave

Use `rsync` or `scp` to copy the data from Master to Slave. If you're syncing the data directly to slave's data directory it's advised to stop the mysqld there.

```
TheMaster$ rsync -avpP -e ssh /path/to/backupdir TheSlave:/path/to/mysql/
```

After data has been copied you can back up the original or previously installed *MySQL datadir* (**NOTE**: Make sure mysqld is shut down before you move the contents of its datadir, or move the snapshot into its datadir.):

```
TheSlave$ mv /path/to/mysql/datadir /path/to/mysql/datadir_bak
```

and move the snapshot from `TheMaster` in its place:

```
TheSlave$ xtrabackup --move-back --target-dir=/path/to/mysql/backupdir
```

After you copy data over, make sure *MySQL* has proper permissions to access them.

```
TheSlave$ chown mysql:mysql /path/to/mysql/datadir
```

In case the ibdata and iblog files are located in different directories outside of the datadir, you will have to put them in their proper place after the logs have been applied.

STEP 3: Configure The Master's MySQL server

Add the appropriate grant in order for slave to be able to connect to master:

```
TheMaster|mysql> GRANT REPLICATION SLAVE ON *.*  TO 'repl'@'$slaveip'
IDENTIFIED BY '$slavepass';
```

Also make sure that firewall rules are correct and that `TheSlave` can connect to `TheMaster`. Test that you can run the mysql client on `TheSlave`, connect to `TheMaster`, and authenticate.

```
TheSlave$ mysql --host=TheMaster --user=repl --password=$slavepass
```

Verify the privileges.

```
mysql> SHOW GRANTS;
```

STEP 4: Configure The Slave's MySQL server

First copy the *my.cnf* file from `TheMaster` to `TheSlave`:

```
TheSlave$ scp user@TheMaster:/etc/mysql/my.cnf /etc/mysql/my.cnf
```

then change the following options in /etc/mysql/my.cnf:

```
server-id=2
```

and start/restart **`mysqld`** on `TheSlave`.

In case you're using init script on Debian based system to start mysqld, be sure that the password for `debian-sys-maint` user has been updated and it's the same as that user's password on the `TheMaster`. Password can be seen and updated in `/etc/mysql/debian.cnf`.

STEP 5: Start the replication

Look at the content of the file `xtrabackup_binlog_info`, it will be something like:

```
TheSlave$ cat /var/lib/mysql/xtrabackup_binlog_info
TheMaster-bin.000001     481
```

Execute the `CHANGE MASTER` statement on a MySQL console and use the username and password you've set up in STEP 3:

```
TheSlave|mysql> CHANGE MASTER TO
                MASTER_HOST='$masterip',
                MASTER_USER='repl',
                MASTER_PASSWORD='$slavepass',
                MASTER_LOG_FILE='TheMaster-bin.000001',
                MASTER_LOG_POS=481;
```

and start the slave:

```
TheSlave|mysql> START SLAVE;
```

STEP 6: Check

You should check that everything went OK with:

```
TheSlave|mysql> SHOW SLAVE STATUS \G
         ...
         Slave_IO_Running: Yes
         Slave_SQL_Running: Yes
         ...
         Seconds_Behind_Master: 13
         ...
```

Both `IO` and `SQL` threads need to be running. The `Seconds_Behind_Master` means the `SQL` currently being executed has a `current_timestamp` of 13 seconds ago. It is an estimation of the lag between `TheMaster` and `TheSlave`. Note that at the beginning, a high value could be shown because `TheSlave` has to "catch up" with `TheMaster`.

Adding more slaves to The Master

You can use this procedure with slight variation to add new slaves to a master. We will use *Percona XtraBackup* to clone an already configured slave. We will continue using the previous scenario for convenience but we will add `TheNewSlave` to the plot.

At `TheSlave`, do a full backup:

```
TheSlave$ xtrabackup --user=yourDBuser --password=MaGiCiGaM \
--backup --slave-info --target-dir=/path/to/backupdir
```

By using the *`xtrabackup --slave-info`* *Percona XtraBackup* creates additional file called `xtrabackup_slave_info`.

Apply the logs:

```
TheSlave$ xtrabackup --prepare --use-memory=2G --target-dir=/path/to/backupdir/
```

Copy the directory from the `TheSlave` to `TheNewSlave` (**NOTE**: Make sure mysqld is shut down on `TheNewSlave` before you copy the contents the snapshot into its *datadir*.):

```
rsync -avprP -e ssh /path/to/backupdir TheNewSlave:/path/to/mysql/datadir
```

Add additional grant on the master:

```
TheMaster|mysql> GRANT REPLICATION SLAVE ON *.*  TO 'repl'@'$newslaveip'
             IDENTIFIED BY '$slavepass';
```

Copy the configuration file from `TheSlave`:

```
TheNEWSlave$ scp user@TheSlave:/etc/mysql/my.cnf /etc/mysql/my.cnf
```

Make sure you change the server-id variable in `/etc/mysql/my.cnf` to 3 and disable the replication on start:

```
skip-slave-start
server-id=3
```

After setting `server_id`, start **`mysqld`**.

Fetch the master_log_file and master_log_pos from the file `xtrabackup_slave_info`, execute the statement for setting up the master and the log file for `The NEW Slave`:

```
TheNEWSlave|mysql> CHANGE MASTER TO
                   MASTER_HOST='$masterip',
                   MASTER_USER='repl',
                   MASTER_PASSWORD='$slavepass',
                   MASTER_LOG_FILE='TheMaster-bin.000001',
                   MASTER_LOG_POS=481;
```

and start the slave:

```
TheNEWSlave|mysql> START SLAVE;
```

If both IO and SQL threads are running when you check the `TheNewSlave`, server is replicating `TheMaster`.

Verifying Backups with replication and pt-checksum

One way to verify if the backup is consistent is by setting up the replication and running pt-table-checksum. This can be used to verify any type of backups, but before setting up replication, backup should be prepared and be able to run (this means that incremental backups should be merged to full backups, encrypted backups decrypted etc.).

Setting up the replication

How to setup a slave for replication in 6 simple steps with Percona XtraBackup guide provides a detailed instructions on how to take the backup and set up the replication.

For checking the backup consistency you can use either the original server where the backup was taken, or another test server created by using a different backup method (such as cold backup, mysqldump or LVM snapshots) as the master server in the replication setup.

Using pt-table-checksum

This tool is part of the *Percona Toolkit*. It performs an online replication consistency check by executing checksum queries on the master, which produces different results on replicas that are inconsistent with the master.

After you confirmed that replication has been set up successfully, you can install or download *pt-table-checksum*. This example shows downloading the latest version of *pt-table-checksum*:

```
$ wget percona.com/get/pt-table-checksum
```

Note: In order for pt-table-checksum to work correctly `libdbd-mysql-perl` will need to be installed on *Debian/Ubuntu* systems or `perl-DBD-MySQL` on *RHEL/CentOS*. If you installed the *percona-toolkit* package from the Percona repositories package manager should install those libraries automatically.

After this command has been run, *pt-table-checksum* will be downloaded to your current working directory.

Running the *pt-table-checksum* on the master will create `percona` database with the `checksums` table which will be replicated to the slaves as well. Example of the *pt-table-checksum* will look like this:

```
$ ./pt-table-checksum
    TS ERRORS  DIFFS     ROWS  CHUNKS SKIPPED    TIME TABLE
    04-30T11:31:50      0      0   633135       8       0   5.400 exampledb.aka_name
04-30T11:31:52      0      0   290859       1       0   2.692 exampledb.aka_title
Checksumming exampledb.user_info:  16% 02:27 remain
Checksumming exampledb.user_info:  34% 01:58 remain
Checksumming exampledb.user_info:  50% 01:29 remain
Checksumming exampledb.user_info:  68% 00:56 remain
Checksumming exampledb.user_info:  86% 00:24 remain
04-30T11:34:38      0      0 22187768     126       0 165.216 exampledb.user_info
04-30T11:38:09      0      0        0       1       0   0.033 mysql.time_zone_name
04-30T11:38:09      0      0        0       1       0   0.052 mysql.time_zone_
↪transition
04-30T11:38:09      0      0        0       1       0   0.054 mysql.time_zone_
↪transition_type
04-30T11:38:09      0      0        8       1       0   0.064 mysql.user
```

If all the values in the `DIFFS` column are 0 that means that backup is consistent with the current setup.

In case backup wasn't consistent *pt-table-checksum* should spot the difference and point to the table that doesn't match. Following example shows adding new user on the backed up slave in order to simulate the inconsistent backup:

```
mysql> grant usage on exampledb.* to exampledb@localhost identified by
↪'thisisnewpassword';
```

If we run the *pt-table-checksum* now difference should be spotted

```
$ ./pt-table-checksum
TS ERRORS  DIFFS     ROWS  CHUNKS SKIPPED    TIME TABLE
04-30T11:31:50      0      0   633135       8       0   5.400 exampledb.aka_name
04-30T11:31:52      0      0   290859       1       0   2.692 exampledb.aka_title
Checksumming exampledb.user_info:  16% 02:27 remain
Checksumming exampledb.user_info:  34% 01:58 remain
Checksumming exampledb.user_info:  50% 01:29 remain
Checksumming exampledb.user_info:  68% 00:56 remain
Checksumming exampledb.user_info:  86% 00:24 remain
04-30T11:34:38      0      0 22187768     126       0 165.216 exampledb.user_info
04-30T11:38:09      0      0        0       1       0   0.033 mysql.time_zone_name
04-30T11:38:09      0      0        0       1       0   0.052 mysql.time_zone_
↪transition
04-30T11:38:09      0      0        0       1       0   0.054 mysql.time_zone_
↪transition_type
04-30T11:38:09      1      0        8       1       0   0.064 mysql.user
```

This output shows that slave and the replica aren't in consistent state and that the difference is in the `mysql.user` table.

More information on different options that pt-table-checksum provides can be found in the *pt-table-checksum* documentation.

How to create a new (or repair a broken) GTID based slave

MySQL 5.6 introduced the new Global Transaction ID (GTID) support in replication. *Percona XtraBackup* automatically stores the `GTID` value in the `xtrabackup_binlog_info` when doing the backup of *MySQL* and *Percona Server for MySQL* 5.7 with the `GTID` mode enabled. This information can be used to create a new (or repair a broken) `GTID` based slave.

STEP 1: Take a backup from any server on the replication environment, master or slave

The following command takes a backup and saves it in the `/data/backups/$TIMESTAMP` folder:

```
$ innobackupex /data/backups/
```

In the destination folder, there will be a file with the name `xtrabackup_binlog_info`. This file contains both binary log coordinates and the `GTID` information.

```
$ cat xtrabackup_binlog_info
mysql-bin.000002    1232        c777888a-b6df-11e2-a604-080027635ef5:1-4
```

That information is also printed by innobackupex after taking the backup:

```
innobackupex: MySQL binlog position: filename 'mysql-bin.000002', position 1232, GTID␣
↪of the last change 'c777888a-b6df-11e2-a604-080027635ef5:1-4'
```

STEP 2: Prepare the backup

The backup will be prepared with the following command:

```
TheMaster$ innobackupex --apply-log /data/backups/$TIMESTAMP/
```

You need to select the path where your snapshot has been taken, for example `/data/backups/2013-05-07_08-33-33`. If everything is ok you should get the same OK message. Now, the transaction logs are applied to the data files, and new ones are created: your data files are ready to be used by the MySQL server.

STEP 3: Move the backup to the destination server

Use **`rsync`** or **`scp`** to copy the data to the destination server. If you are synchronizing the data directly to the already running slave's data directory it is advised to stop the *MySQL* server there.

```
$ rsync -avprP -e ssh /path/to/backupdir/$TIMESTAMP NewSlave:/path/to/mysql/
```

After you copy the data over, make sure *MySQL* has proper permissions to access them.

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

STEP 4: Configure and start replication

Set the gtid_purged variable to the `GTID` from `xtrabackup_binlog_info`. Then, update the information about the master node and, finally, start the slave.

```
# Using the mysql shell
NewSlave > SET SESSION wsrep_on = 0;
NewSlave > RESET MASTER;
NewSlave > SET SESSION wsrep_on = 1;
NewSlave > SET GLOBAL gtid_purged='<gtid_string_found_in_xtrabackup_binlog_info>';
NewSlave > CHANGE MASTER TO
             MASTER_HOST="$masterip",
             MASTER_USER="repl",
             MASTER_PASSWORD="$slavepass",
             MASTER_AUTO_POSITION = 1;
NewSlave > START SLAVE;
```

Note: The example above is applicable to **|PXC|**. The `wsrep_on` variable is set to *0* before resetting the master (`RESET MASTER`). The reason is that **|PXC|** will not allow resetting the master if `wsrep_on=1`.

STEP 5: Check the replication status

Following command will show the slave status:

```
NewSlave > SHOW SLAVE STATUS\G
         [..]
         Slave_IO_Running: Yes
         Slave_SQL_Running: Yes
         [...]
```

```
        Retrieved_Gtid_Set: c777888a-b6df-11e2-a604-080027635ef5:5
        Executed_Gtid_Set: c777888a-b6df-11e2-a604-080027635ef5:1-5
```

We can see that the slave has retrieved a new transaction with number 5, so transactions from 1 to 5 are already on this slave.

That's all, we have created a new slave in our GTID based replication environment.

Auxiliary Guides

Enabling the server to communicate via TCP/IP

Most of the Linux distributions do not enable by default to accept TCP/IP connections from outside in their MySQL or Percona Server packages.

You can check it with netstat on a shell:

```
$ netstat -lnp | grep mysql
tcp          0         0 0.0.0.0:3306 0.0.0.0:* LISTEN 2480/mysqld
unix 2 [ ACC ] STREAM LISTENING 8101 2480/mysqld /tmp/mysql.sock
```

You should check two things:

- there is a line starting with tcp (the server is indeed accepting TCP connections) and
- the first address (0.0.0.0:3306 in this example) is different than 127.0.0.1:3306 (the bind address is not localhost's).

In the first case, the first place to look is the my.cnf file. If you find the option skip-networking, comment it out or just delete it. Also check that *if* the variable bind_address is set, then it shouldn't be set to localhost's but to the host's IP. Then restart the MySQL server and check it again with netstat. If the changes you did had no effect, then you should look at your distribution's startup scripts (like rc.mysqld). You should comment out flags like --skip-networking and/or change the bind-address.

After you get the server listening to remote TCP connections properly, the last thing to do is checking that the port (3306 by default) is indeed open. Check your firewall configurations (iptables -L) and that you are allowing remote hosts on that port (in /etc/hosts.allow).

And we're done! We have a MySQL server running which is able to communicate with the world through TCP/IP.

Privileges and Permissions for Users

We will be referring to "permissions" to the ability of a user to access and perform changes on the relevant parts of the host's filesystem, starting/stopping services and installing software.

By "privileges" we refer to the abilities of a database user to perform different kinds of actions on the database server.

At a system level

There are many ways for checking the permission on a file or directory. For example, ls -ls /path/to/file or stat /path/to/file | grep Access will do the job:

```
$ stat /etc/mysql | grep Access
Access: (0755/drwxr-xr-x)  Uid: (    0/    root)   Gid: (    0/    root)
Access: 2011-05-12 21:19:07.129850437 -0300
$ ls -ld /etc/mysql/my.cnf
-rw-r--r-- 1 root root 4703 Apr  5 06:26 /etc/mysql/my.cnf
```

As in this example, `my.cnf` is owned by `root` and not writable for anyone else. Assuming that you do not have `root` 's password, you can check what permissions you have on this types of files with `sudo -l`:

```
$ sudo -l
Password:
You may run the following commands on this host:
(root) /usr/bin/
(root) NOPASSWD: /etc/init.d/mysqld
(root) NOPASSWD: /bin/vi /etc/mysql/my.cnf
(root) NOPASSWD: /usr/local/bin/top
(root) NOPASSWD: /usr/bin/ls
(root) /bin/tail
```

Being able to execute with `sudo` scripts in `/etc/init.d/`, `/etc/rc.d/` or `/sbin/service` is the ability to start and stop services.

Also, If you can execute the package manager of your distribution, you can install or remove software with it. If not, having `rwx` permission over a directory will let you do a local installation of software by compiling it there. This is a typical situation in many hosting companies' services.

There are other ways for managing permissions, such as using *PolicyKit*, *Extended ACLs* or *SELinux*, which may be preventing or allowing your access. You should check them in that case.

At a database server level

To query the privileges that your database user has been granted, at a console of the server execute:

```
mysql> SHOW GRANTS;
```

or for a particular user with:

```
mysql> SHOW GRANTS FOR 'db-user'@'host';
```

It will display the privileges using the same format as for the GRANT statement.

Note that privileges may vary across versions of the server. To list the exact list of privileges that your server support (and a brief description of them) execute:

```
mysql> SHOW PRIVILEGES;
```

Installing and configuring a SSH server

Many Linux distributions only install the ssh client by default. If you don't have the ssh server installed already, the easiest way of doing it is by using your distribution's packaging system:

```
   ubuntu$ sudo apt-get install openssh-server
archlinux$ sudo pacman -S openssh
```

You may need to take a look at your distribution's documentation or search for a tutorial on the internet to configure it if you haven't done it before.

Some links of them are:

- Debian - http://wiki.debian.org/SSH
- Ubuntu - https://help.ubuntu.com/10.04/serverguide/C/openssh-server.html
- Archlinux - https://wiki.archlinux.org/index.php/SSH
- Fedora - http://docs.fedoraproject.org/en-US/Fedora/12/html/Deployment_Guide/s1-openssh-server-config.html
- CentOS - http://www.centos.org/docs/5/html/Deployment_Guide-en-US/s1-openssh-server-config.html
- RHEL - http://docs.redhat.com/docs/en-US/Red_Hat_Enterprise_Linux/6/html/Deployment_Guide/ch-OpenSSH.html

Assumptions in this section

Most of the times, the context will make the recipe or tutorial understandable. To assure that, a list of the assumptions, names and "things" that will appear in this section is given. At the beginning of each recipe or tutorial they will be specified in order to make it quicker and more practical.

`HOST`

A system with a *MySQL*-based server installed, configured and running. We will assume the following about this system:

- the MySQL server is able to *communicate with others by the standard TCP/IP port*;
- a SSH server is installed and configured - see *here* if it is not;
- you have an user account in the system with the appropriate *permissions* and
- you have a MySQL's user account with appropriate *Connection and Privileges Needed*.

`USER` An user account in the system with shell access and appropriate permissions for the task. A guide for checking them is *here*.

`DB-USER` An user account in the database server with appropriate privileges for the task. A guide for checking them is *here*.

- *Recipes for xtrabackup*
- *Recipes for innobackupex*
- *How-Tos*
- *Auxiliary Guides*

Part VIII

References

CHAPTER

FOURTEEN

KNOWN ISSUES AND LIMITATIONS

There is a number of *Percona XtraBackup* related issues with compressed *InnoDB* tables. These issues result from either server-side bugs, or OS configuration and thus, cannot be fixed on the *Percona XtraBackup* side.

Known issues:

- For *MySQL* or *Percona Server for MySQL* versions 5.1 and 5.5 there are known and unfixed bugs with redo-logging of updates to compressed *InnoDB* tables. For example, internal Oracle bug #16267120 has been fixed only in *MySQL* 5.6.12, but not in 5.1 or 5.5. The bug is about compressed page images not being logged on page reorganization and thus, creating a possibility for recovery process to fail in case a different zlib version is being used when replaying a `MLOG_ZIP_PAGE_REORGANIZE` redo log record.
- For *MySQL* or *Percona Server for MySQL* version 5.6 it is **NOT** recommended to set `innodb_log_compressed_pages=OFF` for servers that use compressed *InnoDB* tables which are backed up with *Percona XtraBackup*. This option makes *InnoDB* recovery (and thus, backup prepare) sensible to `zlib` versions. In case the host where a backup prepare is performed uses a different `zlib` version than the one that was used by the server during runtime, backup prepare may fail due to differences in compression algorithms.
- Backed-up table data could not be recovered if backup was taken while running `OPTIMIZE TABLE` (bug https://bugs.launchpad.net/percona-xtrabackup/+bug/1541763) or `ALTER TABLE ... TABLESPACE` (bug PXB-1360) on that table.
- Compact Backups currently don't work due to bug PXB-372.
- Backup fails with `Error 24: 'Too many open files'`. This usually happens when database being backed up contains large amount of files and *Percona XtraBackup* can't open all of them to create a successful backup. In order to avoid this error the operating system should be configured appropriately so that *Percona XtraBackup* can open all its files. On Linux, this can be done with the `ulimit` command for specific backup session or by editing the `/etc/security/limits.conf` to change it globally (**NOTE**: the maximum possible value that can be set up is `1048576` which is a hard-coded constant in the Linux kernel).

The `xtrabackup` binary has some limitations you should be aware of to ensure that your backups go smoothly and are recoverable.

Limitations:

- The Aria storage engine is part of *MariaDB* and has been integrated in it for many years and Aria table files backup support has been added to **innobackupex** in 2011. The issue is that the engine uses recovery log files and an `aria_log_control` file that are not backed up by **xtrabackup**. As stated in the documentation, starting *MariaDB* without the `maria_log_control` file will mark all the Aria tables as corrupted with this error when doing a `CHECK` on the table: `Table is from another system and must be zerofilled or repaired to be usable on this system`. This means that the Aria tables from an **xtrabackup** backup must be repaired before being usable (this could be quite long depending on the size of the table). Another option is `aria_chk --zerofil table` on all Aria tables present on the backup after the prepare phase.

- If the `xtrabackup_logfile` is larger than 4GB, the `--prepare` step will fail on 32-bit versions of `xtrabackup`.
- `xtrabackup` doesn't understand the very old `--set-variable` `my.cnf` syntax that MySQL uses. See *Configuring xtrabackup*.

CHAPTER

FIFTEEN

FREQUENTLY ASKED QUESTIONS

Do I need an InnoDB Hot Backup license to use Percona XtraBackup?

No. Although `innobackupex` is derived from the same GPL and open-source wrapper script that InnoDB Hot Backup uses, it does not execute `ibbackup`, and the `xtrabackup` binary does not execute or link to `ibbackup`. You can use *Percona XtraBackup* without any license; it is completely separate from InnoDB Hot Backup.

What's the difference between `innobackupex` and `innobackup`?

Because **innobackupex** is a patched version of *Oracle* 's **innobackup** script (now renamed to **mysqlbackup**), it is quite similar in some ways, and familiarity with innobackup might be helpful.

Aside from the options for specific features of **innobackupex**, the main differences are:

- printing to `STDERR` instead of `STDOUT` (which enables the *`innobackupex --stream`* option),
- the configuration file - *my.cnf* - is detected automatically (or set with *`innobackupex --defaults-file`*) instead of the mandotory first argument,
- and defaults to **xtrabackup** as binary to use in the *`innobackupex --ibbackup`*.

See *The innobackupex Option Reference* for more details.

Are you aware of any web-based backup management tools (commercial or not) built around *Percona XtraBackup*?

Zmanda Recovery Manager is a commercial tool that uses *Percona XtraBackup* for Non-Blocking Backups:

> *"ZRM provides support for non-blocking backups of MySQL using |Percona XtraBackup|. ZRM with |Percona XtraBackup| provides resource utilization management by providing throttling based on the number of IO operations per second. |Percona XtraBackup| based backups also allow for table level recovery even though the backup was done at the database level (needs the recovery database server to be |Percona Server| with XtraDB)."*

`xtrabackup` binary fails with a floating point exception

In most of the cases this is due to not having install the required libraries (and version) by **`xtrabackup`**. Installing the *GCC* suite with the supporting libraries and recompiling **`xtrabackup`** will solve the issue. See *Compiling and Installing from Source Code* for instructions on the procedure.

How xtrabackup handles the ibdata/ib_log files on restore if they aren't in

mysql datadir?

In case the `ibdata` and `ib_log` files are located in different directories outside of the datadir, you will have to put them in their proper place after the logs have been applied.

Backup fails with Error 24: 'Too many open files'

This usually happens when database being backed up contains large amount of files and *Percona XtraBackup* can't open all of them to create a successful backup. In order to avoid this error the operating system should be configured appropriately so that *Percona XtraBackup* can open all its files. On Linux, this can be done with the `ulimit` command for specific backup session or by editing the `/etc/security/limits.conf` to change it globally (**NOTE**: the maximum possible value that can be set up is `1048576` which is a hard-coded constant in the Linux kernel).

How to deal with skipping of redo logs for DDL operations?

To prevent creating corrupted backups when running DDL operations, Percona XtraBackup aborts if it detects that redo logging is disabled. In this case, the following error is printed:

```
[FATAL] InnoDB: An optimized (without redo logging) DDL operation has been performed.␣
↪All modified pages may not have been flushed to the disk yet.
Percona XtraBackup will not be able to take a consistent backup. Retry the backup␣
↪operation.
```

Note: Redo logging is disabled during a sorted index build

To avoid this error, Percona XtraBackup can use metadata locks on tables while they are copied:

- To block all DDL operations, use the *`xtrabackup --lock-ddl`* option that issues `LOCK TABLES FOR BACKUP`.
- If `LOCK TABLES FOR BACKUP` is not supported, you can block DDL for each table before XtraBackup starts to copy it and until the backup is completed using the *`xtrabackup --lock-ddl-per-table`* option.

CHAPTER

SIXTEEN

PERCONA XTRABACKUP 2.4 RELEASE NOTES

Percona XtraBackup 2.4.20

Date April 14, 2020

Installation *Installing Percona XtraBackup 2.4*

Downloads are available from our download site and from *apt* and *yum* repositories.

Percona XtraBackup enables MySQL backups without blocking user queries, making it ideal for companies with large data sets and mission-critical applications that cannot tolerate long periods of downtime. Offered free as an open source solution, it drives down backup costs while providing unique features for *MySQL* backups.

All Percona software is open-source and free.

This release fixes security vulnerability *CVE-2020-10997*

Percona XtraBackup 2.4.19

Date March 25, 2020

Installation *Installing Percona XtraBackup 2.4*

Downloads are available from our download site and from *apt* and *yum* repositories.

Percona XtraBackup enables MySQL backups without blocking user queries, making it ideal for companies with large data sets and mission-critical applications that cannot tolerate long periods of downtime. Offered free as an open source solution, it drives down backup costs while providing unique features for *MySQL* backups.

All Percona software is open-source and free.

Bugs Fixed

- PXB-1982: The *history* table showed a wrong value for `lock_time`.

Percona XtraBackup 2.4.18

Percona is glad to announce the release of *Percona XtraBackup* 2.4.18 on December 16, 2019. Downloads are available from our download site and from *apt* and *yum* repositories.

All Percona software is open-source and free.

Bugs Fixed

- Sometime between December 3rd and December 10th, a change was introduced in AWS (Amazon Web Services) that caused an incompatibility with our *Percona XtraBackup* `xbcloud` utility. Bug fixed PXB-1978.

Percona XtraBackup 2.4.17

Percona is glad to announce the release of *Percona XtraBackup* 2.4.17 on December 9, 2019. Downloads are available from our download site and from *apt* and *yum* repositories.

All Percona software is open-source and free.

Bugs Fixed

- *Percona XtraBackup* could crash when making a backup for Percona Server 5.7.28-31 where the tablespace encryption was used. Bug fixed PXB-1968.

Percona XtraBackup 2.4.16

Percona is glad to announce the release of *Percona XtraBackup* 2.4.16 on November 4, 2019. Downloads are available from our download site and from *apt* and *yum* repositories.

All Percona software is open-source and free.

Improvements

- Two options (–backup-lock-timeout and –backup-lock-retry-count) were added to enable the configuring of the timeout for acquiring metadata locks in `FLUSH TABLES WITH READ LOCK`, `LOCK TABLE FOR BACKUP`, and `LOCK BINLOG FOR BACKUP` statements. More information in PXB-1914

Bugs Fixed

- *Percona XtraBackup* was not able to connect to the database when the password was specified along with the transition-key parameter. Bug fixed PXB-1902.
- In some cases, *Percona XtraBackup* stuck with redo log corruption when master key is rotated. Bug fixed PXB-1903.
- In rare cases, when both full and incremental backups were made before MySQL flushed the first page of the encrypted tablespace, *Percona XtraBackup* could crash during the incremental backup prepare for the tablespace encryption. Bug fixed PXB-1894.
- An encrypted table could not be restored when ADD/DROP INDEX was run on the table. Bug fixed PXB-1905.
- In some cases `xtrabackup --prepare` could fail to decrypt a table but reported that the operation *completed ok*. Bug fixed PXB-1936.

Percona XtraBackup 2.4.15

Percona is glad to announce the release of *Percona XtraBackup* 2.4.15 on July 10, 2019. Downloads are available from our download site and from *apt* and *yum* repositories.

All Percona software is open-source and free.

Bugs Fixed

- When the *encrypted tablespaces* feature was enabled, encrypted and compressed tables were not usable on the joiner node (Percona XtraDB Cluster) via SST (State Snapshot Transfer) with the `xtrabackup-v2` method. Bug fixed PXB-1867.
- `xbcloud` did not update date related fields of the HTTP header when retrying a request. Bug fixed PXB-1874.
- `xbcloud` did not retry to send the request after receiving the HTTP 408 error (request timeout). Bug fixed PXB-1875.
- If the user tried to merge an already prepared incremental backup, a misleading error was produced without informing that incremental backups may not be used twice. Bug fixed PXB-1862.
- `xbcloud` could crash with the *Swift* storage when project options were not included. Bug fixed PXB-1844.
- `xtrabackup` did not accept decimal fractions as values of the `innodb_max_dirty_pages_pct` option. Bug fixed PXB-1807.

Other bugs fixed: PXB-1850, PXB-1879, PXB-1887, PXB-1888, PXB-1890.

Percona XtraBackup 2.4.14

Percona is glad to announce the release of *Percona XtraBackup* 2.4.14 on May 1, 2019. Downloads are available from our download site and from *apt* and *yum* repositories.

Percona XtraBackup 2.4.14 enables saving backups to an Amazon S3 storage when using *xbcloud*. The following example demonstrates how to use an Amazon S3 storage to make a full backup.

```
$ xtrabackup --backup --stream=xbstream --extra-lsndir=/tmp --target-dir=/tmp | \
xbcloud put --storage=s3 \
--s3-endpoint='s3.amazonaws.com' \
--s3-access-key='YOUR-ACCESSKEYID' \
--s3-secret-key='YOUR-SECRETACCESSKEY' \
--s3-bucket='mysql_backups'
--parallel=10 \
${date -I}-full_backup
```

All Percona software is open-source and free.

New Features

- Amazon S3 is now supported in *xbcloud*. More information in PXB-1813.

Bugs Fixed

- When the row format was changed during the backup, **xtrabackup** could crash during the incremental prepare stage. Bug fixed PXB-1824.
- If compressed InnoDB undo tablespaces were not removed beforehand, the incremental backup could crash at the prepare stage. Bug fixed PXB-1552.

Other bugs fixed: PXB-1771, PXB-1809, PXB-1837.

Percona XtraBackup 2.4.13

Percona is glad to announce the release of *Percona XtraBackup* 2.4.13 on January 18, 2018. Downloads are available from our download site and from *apt* and *yum* repositories.

All Percona software is open-source and free.

Improvements and New Features

- PXB-1548: *Percona XtraBackup* enables updating the ib_buffer_pool file with the latest pages present in the buffer pool by setting the `xtrabackup --dump-innodb-buffer-pool` option to *ON*. Thanks to Marcelo Altmann for contribution.

Bus Fixed

- `xtrabackup` did not delete missing tables from the partial backup which led to error messages logged by the server on startup. Bug fixed PXB-1536.
- The `--history` option did not work when autocommit was disabled. Bug fixed PXB-1569.
- `xtrabackup` could fail to backup encrypted tablespace when it was recently created or altered. Bug fixed PXB-1648.
- When the `--throttle` option was used, the applied value was different from the one specified by the user (off by one error). Bug fixed PXB-1668.
- It was not allowed for MTS (multi-threaded slaves) without GTID to be backed up with `--safe-slave-backup`. Bug fixed PXB-1672.
- *Percona XtraBackup* could crash when the `ALTER TABLE ... TRUNCATE PARTITION` command was run during a backup without locking DDL. Bug fixed PXB-1679.
- `xbcrypt` could display an assertion failure and generated core if the required parameters are missing. Bug fixed PXB-1683.
- Using `--lock-ddl-per-table` caused the server to scan all records of partitioned tables which could lead to the "out of memory" error. Bugs fixed PXB-1691 and PXB-1698.
- *xtrabackup –prepare* could hang while performing insert buffer merge. Bug fixed PXB-1704.
- Incremental backups did not update `xtrabackup_binlog_info` with `--binlog-info=lockless`. Bug fixed PXB-1711.

Other bugs fixed: PXB-1570, PXB-1609, PXB-1632

Percona XtraBackup 2.4.12

Percona is glad to announce the release of *Percona XtraBackup* 2.4.12 on June 22, 2018. Downloads are available from our download site and from *apt* and *yum* repositories.

All Percona software is open-source and free.

New features and improvements

- *Percona XtraBackup* now prints used arguments to standard output. Bug fixed PXB-1494.

Bugs fixed

- `xtrabackup --copy-back` didn't read which encryption plugin to use from `plugin-load` setting of the `my.cnf` configuration file. Bug fixed PXB-1544.
- xbstream was exiting with zero return code when it failed to create one or more target files instead of returning error code 1. Bug fixed PXB-1542.
- Meeting a zero sized keyring file, *Percona XtraBackup* was removing and immediately recreating it, which could affect external software noticing this file had undergo manipulations. Bug fixed PXB-1540.
- `xtrabackup_checkpoints` files were encrypted during a backup, which caused additional difficulties to take incremental backups. Bug fixed PXB-202.

Other bugs fixed: PXB-1526 "Test kill_long_selects.sh failing with MySQL 5.7.21".

Percona XtraBackup 2.4.11

Percona is glad to announce the release of *Percona XtraBackup* 2.4.11 on April 23, 2018. Downloads are available from our download site and from *apt* and *yum* repositories.

All Percona software is open-source and free.

New features and improvements

- The support of the *Percona Server for MySQL* encrypted general tablespaces was implemented in this version of *Percona XtraBackup*. Issue fixed PXB-1513.
- *Percona XtraBackup* is now able to backup encrypted *Percona Server for MySQL* instances which are using keyring_vault plugin. Issue fixed PXB-1514.

Percona XtraBackup 2.4.10

Percona is glad to announce the release of *Percona XtraBackup* 2.4.10 on March 30, 2018. Downloads are available from our download site and from *apt* and *yum* repositories.

This release is based on MySQL 5.7.19 and is the current GA (Generally Available) stable release in the 2.4 series.

Starting from now, *Percona XtraBackup* issue tracking system was moved from launchpad to JIRA. All Percona software is open-source and free.

Bugs fixed

- xbcrypt with `--encrypt-key-file` option was failing due to regression in *Percona XtraBackup* 2.4.9. Bug fixed PXB-518.
- Simultaneous usage of both `--lock-ddl` and `--lock-ddl-per-table` options caused *Percona XtraBackup* lock with the backup process never completed. Bug fixed PXB-792.
- Compilation under Mac OS X was broken. Bug fixed PXB-796.
- A regression of the maximum number of pending reads and the unnoticed earlier possibility of a pending reads related deadlock caused *Percona XtraBackup* to stuck in prepare stage. Bug fixed: PXB-1467.
- *Percona XtraBackup* skipped tablespaces with corrupted first page instead of aborting the backup. Bug fixed PXB-1497.

Other bugs fixed: PXB-513.

Percona XtraBackup 2.4.9

Percona is glad to announce the release of *Percona XtraBackup* 2.4.9 on November 29th 2017. Downloads are available from our download site and from *apt* and *yum* repositories.

This release is the current GA (Generally Available) stable release in the 2.4 series.

New Features

Percona XtraBackup packages are now available for *Ubuntu 17.10 (Artful)*.

xbcrypt now has an ability to decrypt files in parallel by specifying the number of threads with the `xtrabackup --encrypt-threads` option.

`xtrabackup --copy-back` option can now be used with `xtrabackup --parallel` option to copy the user data files in parallel (redo logs and system tablespaces are copied in the main thread).

Bugs fixed

Percona XtraBackup would fail to backup large databases on 32-bit platforms. Bug fixed PXB-481.

Percona XtraBackup failed to build with *GCC 7*. Bug fixed PXB-502.

Percona XtraBackup would hang during the prepare phase if there was not enough room in log buffer to accommodate checkpoint information at the end of the crash recovery process. Bug fixed PXB-506.

When backup was streamed in tar format with with the `xtrabackup --slave-info` option output file `xtrabackup_slave_info` did not contain the slave information. Bug fixed PXB-507.

If `xtrabackup --slave-info` was used while backing up 5.7 instances, the master binary log coordinates were not properly displayed in the logs. Bug fixed PXB-508.

`innobackupex --slave-info` would report a single `m` instead of `slave info` in the standard output. Bug fixed PXB-510.

Percona XtraBackup would crash while preparing the 5.5 backup with `utf8_general50_ci` collation. Bug fixed PXB-748 (*Fungo Wang*).

Percona XtraBackup would crash if `xtrabackup --throttle` was used while preparing backups. Fixed by making this option available only during the backup process. Bug fixed PXB-789.

Percona XtraBackup could get stuck if backups are taken with `xtrabackup --safe-slave-backup` option, while there were long running queries. Bug fixed PXB-1039.

Other bugs fixed: PXB-250, PXB-511, and PXB-512.

Percona XtraBackup 2.4.8

Percona is glad to announce the release of *Percona XtraBackup* 2.4.8 on July 24th 2017. Downloads are available from our download site and from *apt* and *yum* repositories.

This release is the current GA (Generally Available) stable release in the 2.4 series.

New Features

To avoid issues with *MySQL* 5.7 skipping redo log for DDL *Percona XtraBackup* has implemented three new options (`xtrabackup --lock-ddl`, `xtrabackup --lock-ddl-timeout`, `xtrabackup --lock-ddl-per-table`) that can be used to place MDL locks on tables while they are copied.

New `xtrabackup --check-privileges` option has been implemented that can be used to check if *Percona XtraBackup* has all *required privileges* to perform the backup.

Bugs fixed

xtrabackup would hang with `Waiting for master thread to be suspended` message when backup was being prepared. Bug fixed PXB-499.

xtrabackup would fail to prepare the backup with `6th page is not initialized` message in case server didn't properly initialize the page. Bug fixed PXB-500.

xbstream could run out of file descriptors while extracting the backup which contains many tables. Bug fixed PXB-503

When a table was created with the `DATA DIRECTORY` option **xtrabackup** would back up the `.frm` and `.isl` files, but not the `.ibd` file. Due to the missing `.ibd` files backup then could not be restored. Bug fixed PXB-504.

Percona XtraBackup incorrectly determined use of `master_auto_postion` on a slave, and thus generated invalid `xtrabackup_slave_info` file. Bug fixed PXB-505.

Percona XtraBackup will now print a warning if it encounters unsupported storage engine. Bug fixed PXB-713.

Percona XtraBackup would crash while backing up MariaDB 10.2.x with `--ftwrl-*` options. Bug fixed PXB-790.

`xtrabackup --slave-info` didn't write the correct information into `xtrabackup_slave_info` file when multi-source replication was used. Bug fixed PXB-1022.

Along with `xtrabackup_checkpints` file, **xtrabackup** now copies `xtrabackup_info` file into directory specified by `xtrabackup --extra-lsndir` option. Bug fixed PXB-1026.

GTID position was not recorded when `xtrabackup --binlog-info` option was set to `AUTO`. Bug fixed PXB-1030.

Percona XtraBackup 2.4.7-2

Percona is glad to announce the release of *Percona XtraBackup* 2.4.7-2 on May 29th 2017. Downloads are available from our download site and from *apt* and *yum* repositories.

This release is the current GA (Generally Available) stable release in the 2.4 series.

Bugs fixed

Fixed build failure on Debian 9.0 (*Stretch*). Bug fixed PXB-501.

Percona XtraBackup 2.4.7

Percona is glad to announce the release of *Percona XtraBackup* 2.4.7 on April 17th 2017. Downloads are available from our download site and from *apt* and *yum* repositories.

This release is the current GA (Generally Available) stable release in the 2.4 series.

New Features

Percona XtraBackup now uses hardware accelerated implementation of `crc32` where it is supported.

Percona XtraBackup has implemented new options: *`xtrabackup --tables-exclude`* and *`xtrabackup --databases-exclude`* that work similar to *`xtrabackup --tables`* and *`xtrabackup --databases`* options, but exclude given names/paths from backup.

The xbstream binary now supports parallel extraction with the `--parallel` option.

The xbstream binary now supports following new options: `--decrypt`, `--encrypt-threads`, `--encrypt-key`, and `--encrypt-key-file`. When `--decrypt` option is specified *xbstream* will automatically decrypt encrypted files when extracting input stream. Either `--encrypt-key` or `--encrypt-key-file` options must be specified to provide encryption key, but not both. Option `--encrypt-threads` specifies the number of worker threads doing the encryption, default is `1`.

Bugs fixed

Backups were missing `*.isl` files for general tablespace. Bug fixed PXB-494.

In 5.7 *MySQL* changed default checksum algorithm to `crc32`, while **`xtrabackup`** was using `innodb`. This caused **`xtrabackup`** to perform extra checksum calculations which were not needed. Bug fixed PXB-495.

For system tablespaces consisting of multiple files **`xtrabackup`** updated LSN only in first file. This caused *MySQL* versions lower than 5.7 to fail on startup. Bug fixed PXB-498.

`xtrabackup --export` can now export tables that have more than 31 index. Bug fixed PXB-58.

`Unrecognized character \x01; marked by <-- HERE` message could be seen if backups were taken with the version check enabled. Bug fixed PXB-944.

Percona XtraBackup 2.4.6

Percona is glad to announce the release of *Percona XtraBackup* 2.4.6 on February 22nd 2017. Downloads are available from our download site and from *apt* and *yum* repositories.

This release is the GA (Generally Available) stable release in the 2.4 series.

New features

Percona XtraBackup has implemented new `xtrabackup --remove-original` option that can be used to remove the encrypted and compressed files once they've been decrypted/decompressed.

Bugs Fixed

xtrabackup was using username set for server in a configuration file even if a different user was defined in the users configuration file. Bug fixed PXB-463.

Incremental backups did not include `xtrabackup_binlog_info` and `xtrabackup_galera_info` files. Bug fixed PXB-489.

In case a warning was written to stout instead of stderr during the streaming backup, it could cause assertion in the *xbstream*. Bug fixed PXB-491.

`xtrabackup --move-back` did not always restore out-of-datadir tablespaces to their original directories. Bug fixed PXB-492.

innobackupex and **xtrabackup** scripts were showing the password in the **ps** output when it was passed as a command line argument. Bug fixed PXB-585

Incremental backup would fail with path like `~/backup/inc_1` because **xtrabackup** didn't properly expand tilde. Bug fixed PXB-775.

Fixed missing dependency check for `perl(Digest::MD5)` in rpm packages. Bug fixed PXB-777.

Percona XtraBackup now supports `-H`, `-h`, `-u` and `-p` shortcuts for `--hostname`, `--datadir`, `--user` and `--password` respectively. Bugs fixed PXB-947 and PXB-1032.

[UPDATE 2016-02-28]: New packages have been pushed to repositories with incremented package version to address the bug PXB-497.

Other bugs fixed: PXB-945.

Percona XtraBackup 2.4.5

Percona is glad to announce the release of *Percona XtraBackup* 2.4.5 on November 29th 2016. Downloads are available from our download site and from *apt* and *yum* repositories.

This release is the GA (Generally Available) stable release in the 2.4 series.

New features

Percona XtraBackup now supports `SHA256` passwords. Using the `SHA256` algorithm requires either SSL encrypted connection, or using public key encryption for password exchange which is only available when both client and server are linked with OpenSSL.

Percona XtraBackup now supports Command Options for Secure Connections.

NOTE: Due to *xbcrypt* format changes, backups encrypted with this *Percona XtraBackup* version will not be recoverable by older versions.

Bugs Fixed

Percona XtraBackup would crash while preparing the backup, during the shutdown, when master thread was performing checkpoint and purge thread was expecting that all other threads completed or were idle. Bug fixed PXB-483.

Safe slave backup algorithm performed too short delays between retries which could cause backups to fail on a busy servers. Bug fixed PXB-484.

Percona XtraBackup didn't check the logblock checksums. Bug fixed PXB-485.

Fixed new compilation warnings with GCC 6. Bug fixed PXB-487.

xbcrypt was not setting the Initialization Vector (IV) correctly (and thus is was not using an IV). This was causing the same ciphertext to be generated across different runs (for the same message/same key). The IV provides the extra randomness to ensure that the same ciphertext is not generated across runs. Bug fixed PXB-490.

`target-dir` was no longer relative to current directory but to `datadir` instead. Bug fixed PXB-760.

Backup would still succeed even if **`xtrabackup`** would fail to write the metadata. Bug fixed PXB-763.

xbcloud now supports EMC ECS Swift API Authorization requests. Bugs fixed PXB-769 and PXB-770 (*Txomin Barturen*).

Some older versions of *MySQL* did not bother to initialize page type field for pages which are not index pages (see upstream #76262 for more information). Having this page type uninitialized could cause **`xtrabackup`** to crash on prepare. Bug fixed PXB-772.

Percona XtraBackup would fail to backup *MariaDB* 10.2 with the `unsupported server version` error message. Bug fixed PXB-1027.

Fixed misleading error message about missing metadata. Bug fixed PXB-752.

Backing up with an SSL user didn't work correctly. Bug fixed PXB-750.

Other bugs fixed: PXB-486, PXB-771, PXB-773, and PXB-774.

Percona XtraBackup 2.4.4

Percona is glad to announce the release of *Percona XtraBackup* 2.4.4 on July 25th 2016. Downloads are available from our download site and from *apt* and *yum* repositories.

This release is the GA (Generally Available) stable release in the 2.4 series.

New features

Percona XtraBackup has been rebased on *MySQL* 5.7.13.

Bugs Fixed

Percona XtraBackup reported the difference in the actual size of the system tablespace and the size which was stored in the tablespace header. This check is now skipped for tablespaces with autoextend support. Bug fixed PXB-462.

Because *Percona Server for MySQL* 5.5 and *MySQL* 5.6 store the LSN offset for large log files at different places inside the redo log header, *Percona XtraBackup* was trying to guess which offset is better to use by trying to read from each one and compare the log block numbers and assert `lsn_chosen == 1` when both LSNs looked correct, but they were different. Fixed by improving the server detection. Bug fixed PXB-473.

Percona XtraBackup didn't correctly detect when tables were both compressed and encrypted. Bug fixed PXB-477.

Percona XtraBackup would crash if the keyring file was empty. Bug fixed PXB-479.

Backup couldn't be prepared when the size in cache didn't match the physical size. Bug fixed PXB-482.

Free Software Foundation address in copyright notices was outdated. Bug fixed PXB-663.

Backup process would fail if the `datadir` specified on the command-line was not the same as one that is reported by the server. *Percona XtraBackup* now allows the `datadir` from `my.cnf` override the one from `SHOW VARIABLES`. **`xtrabackup`** will print a warning that they don't match, but continue. Bug fixed PXB-741.

With upstream change of maximum page size from 16K to 64K, the size of incremental buffer became 1G. Which increased the requirement to 1G of RAM in order to prepare the backup. While in fact there is no need to allocate such a large buffer for smaller pages. Bug fixed PXB-753.

Backup process would fail on *MariaDB* Galera cluster operating in GTID mode if binary logs were in non-standard directory. Bug fixed PXB-936.

Other bugs fixed: PXB-755, PXB-756, and PXB-759.

Percona XtraBackup 2.4.3

Percona is glad to announce the release of *Percona XtraBackup* 2.4.3 on May 23rd 2016. Downloads are available from our download site and from *apt* and *yum* repositories.

This release is the GA (Generally Available) stable release in the 2.4 series.

New features

Percona XtraBackup has implemented new *`xtrabackup --reencrypt-for-server-id`* option. Using this option allows users to start the server instance with different `server_id` from the one the encrypted backup was taken from, like a replication slave or a galera node. When this option is used, **`xtrabackup`** will, as a prepare step, generate a new master key with ID based on the new `server_id`, store it into keyring file and re-encrypt the tablespace keys inside of tablespace headers.

Bugs Fixed

Running DDL statements on *Percona Server for MySQL* 5.7 during the backup process could in some cases lead to failure while preparing the backup. Bug fixed PXB-247.

MySQL 5.7 can sometimes skip redo logging when creating an index. If such `ALTER TABLE` is being issued during the backup, the backup would be inconsistent. **xtrabackup** will now abort with error message if such `ALTER TABLE` has been done during the backup. Bug fixed PXB-249.

`.ibd` files for remote tablespaces were not copied back to original location pointed by the `.isl` files. Bug fixed PXB-466.

When called with insufficient parameters, like specifying the empty *xtrabackup --defaults-file* option, *Percona XtraBackup* could crash. Bug fixed PXB-471.

Documentation states that the default value for *xtrabackup --ftwrl-wait-query-type* is `all`, however it was `update`. Changed the default value to reflect the documentation. Bug fixed PXB-472.

When *xtrabackup --keyring-file-data* option was specified, but no keyring file was found, **xtrabackup** would create an empty one instead of reporting an error. Bug fixed PXB-476.

If `ALTER INSTANCE ROTATE INNODB MASTER KEY` was run at same time when *xtrabackup --backup* was bootstrapping it could catch a moment when the key was not written into the keyring file yet and **xtrabackup** would overwrite the keyring with the old copy of a keyring, so the new key would be lost. Bug fixed PXB-478.

Output of *xtrabackup --slave-info* option was missing an apostrophe. Bug fixed PXB-940.

Percona XtraBackup 2.4.2

Percona is glad to announce the release of *Percona XtraBackup* 2.4.2 on April 1st 2016. Downloads are available from our download site and from *apt* and *yum* repositories.

This release is the GA (Generally Available) stable release in the 2.4 series.

New features

Percona XtraBackup has *implemented support* for InnoDB tablespace encryption.

Percona XtraBackup has been rebased on *MySQL* 5.7.11.

Bugs Fixed

When backup was taken on *MariaDB* 10 with GTID enabled, *Percona XtraBackup* didn't store `gtid_slave_pos` in `xtrabackup_slave_info` but logged it only to `STDERR`. Bug fixed PXB-715.

Backup process would fail if *xtrabackup --throttle* option was used. Bug fixed PXB-465.

Percona XtraBackup 2.4.1

Percona is glad to announce the release of *Percona XtraBackup* 2.4.1 on February 16th 2016. Downloads are available from our download site and from *apt* and *yum* repositories.

This release is the first GA (Generally Available) stable release in the 2.4 series.

This release contains all the features and bug fixes in Percona XtraBackup 2.3.3, plus the following:

New features

Percona XtraBackup has implemented basic support for *MySQL* 5.7 and *Percona Server for MySQL* 5.7.

Bugs Fixed

Percona XtraBackup didn't respect `innodb_log_file_size` variable stored in `backup-my.cnf`. Bug fixed PXB-450.

If server would run out of space while backups were taken with *`innobackupex --rsync`* option backup process would fail but **`innobackupex`** would still complete with `completed OK!` message. Bug fixed PXB-459.

Percona XtraBackup was silently skipping extra arguments. Bug fixed PXB-747 (*Fungo Wang*).

Other bugs fixed: PXB-1368 and 1363.

Percona XtraBackup 2.4.0-rc1

Percona is glad to announce the release of *Percona XtraBackup* 2.4.0-rc1 on February 8th 2016. Downloads are available from our download site and from *apt* and *yum* repositories.

This is a **Release Candidate** quality release and it is not intended for production. If you want a high quality, Generally Available release, the current Stable version should be used (currently 2.3.3 in the 2.3 series at the time of writing).

New features

Percona XtraBackup has implemented basic support for *MySQL* 5.7 and *Percona Server for MySQL* 5.7.

Known Issues

Backed-up table data could not be recovered if backup was taken while running `OPTIMIZE TABLE` (bug PXB-1360) or `ALTER TABLE ... TABLESPACE` (bug PXB-1360) on that table.

Compact Backups currently don't work due to bug PXB-372.

CHAPTER

SEVENTEEN

GLOSSARY

UUID Universally Unique IDentifier which uniquely identifies the state and the sequence of changes node undergoes. 128-bit UUID is a classic DCE UUID Version 1 (based on current time and MAC address). Although in theory this UUID could be generated based on the real MAC-address, in the Galera it is always (without exception) based on the generated pseudo-random addresses ("locally administered" bit in the node address (in the UUID structure) is always equal to unity).

Complete structure of the 128-bit UUID field and explanation for its generation are as follows:

From	To	Length	Content
0	31	32	Bits 0-31 of Coordinated Universal Time (UTC) as a count of 100-nanosecond intervals since 00:00:00.00, 15 October 1582, encoded as big-endian 32-bit number.
32	47	16	Bits 32-47 of UTC as a count of 100-nanosecond intervals since 00:00:00.00, 15 October 1582, encoded as big-endian 16-bit number.
48	59	12	Bits 48-59 of UTC as a count of 100-nanosecond intervals since 00:00:00.00, 15 October 1582, encoded as big-endian 16-bit number.
60	63	4	UUID version number: always equal to 1 (DCE UUID).
64	69	6	most-significants bits of random number, which generated from the server process PID and Coordinated Universal Time (UTC) as a count of 100-nanosecond intervals since 00:00:00.00, 15 October 1582.
70	71	2	UID variant: always equal to binary 10 (DCE variant).
72	79	8	8 least-significant bits of random number, which generated from the server process PID and Coordinated Universal Time (UTC) as a count of 100-nanosecond intervals since 00:00:00.00, 15 October 1582.
80	80	1	Random bit ("unique node identifier").
81	81	1	Always equal to the one ("locally administered MAC address").
82	127	46	Random bits ("unique node identifier"): readed from the `/dev/urandom` or (if `/dev/urandom` is unavailable) generated based on the server process PID, current time and bits of the default "zero node identifier" (entropy data).

LSN Each InnoDB page (usually 16kb in size) contains a log sequence number, or LSN. The LSN is the system version number for the entire database. Each page's LSN shows how recently it was changed.

innodb_file_per_table By default, all InnoDB tables and indexes are stored in the system tablespace on one file. This option causes the server to create one tablespace file per table. To enable it, set it on your configuration file,

```
[mysqld]
innodb_file_per_table
```

or start the server with `--innodb_file_per_table`.

innodb_expand_import This feature of *Percona Server for MySQL* implements the ability to import arbitrary *.ibd* files exported using the *Percona XtraBackup* `xtrabackup --export` option.

See the the full documentation for more information.

innodb_data_home_dir The directory (relative to *datadir*) where the database server stores the files in a shared tablespace setup. This option does not affect the location of *innodb_file_per_table*. For example:

```
[mysqld]
innodb_data_home_dir = ./
```

innodb_data_file_path Specifies the names, sizes and location of shared tablespace files:

```
[mysqld]
innodb_data_file_path=ibdata1:50M;ibdata2:50M:autoextend
```

innodb_log_group_home_dir Specifies the location of the *InnoDB* log files:

```
[mysqld]
innodb_log_group_home=/var/lib/mysql
```

innodb_buffer_pool_size The size in bytes of the memory buffer to cache data and indexes of *InnoDB*'s tables. This aims to reduce disk access to provide better performance. By default:

```
[mysqld]
innodb_buffer_pool_size=8MB
```

InnoDB Storage engine which provides ACID-compliant transactions and foreign key support, among others improvements over *MyISAM*. It is the default engine for *MySQL* as of the 5.5 series.

MyISAM Previous default storage engine for *MySQL* for versions prior to 5.5. It doesn't fully support transactions but in some scenarios may be faster than *InnoDB*. Each table is stored on disk in 3 files: *.frm*, *.MYD*, *.MYI*.

XtraDB *Percona XtraDB* is an enhanced version of the InnoDB storage engine, designed to better scale on modern hardware, and including a variety of other features useful in high performance environments. It is fully backwards compatible, and so can be used as a drop-in replacement for standard InnoDB. More information here.

my.cnf This file refers to the database server's main configuration file. Most Linux distributions place it as `/etc/mysql/my.cnf` or `/etc/my.cnf`, but the location and name depends on the particular installation. Note that this is not the only way of configuring the server, some systems does not have one even and rely on the command options to start the server and its defaults values.

datadir The directory in which the database server stores its databases. Most Linux distribution use `/var/lib/mysql` by default.

xbcrypt To support encryption and decryption of the backups, a new tool xbcrypt was introduced to *Percona XtraBackup*. This utility has been modeled after The xbstream binary to perform encryption and decryption outside of *Percona XtraBackup*.

xbstream To support simultaneous compression and streaming, a new custom streaming format called xbstream was introduced to *Percona XtraBackup* in addition to the TAR format.

ibdata Default prefix for tablespace files, e.g. `ibdata1` is a 10MB auto-extensible file that *MySQL* creates for the shared tablespace by default.

.frm For each table, the server will create a file with the `.frm` extension containing the table definition (for all storage engines).

.ibd On a multiple tablespace setup (*innodb_file_per_table* enabled), *MySQL* will store each newly created table on a file with a `.ibd` extension.

.MYD Each *MyISAM* table has `.MYD` (MYData) file which contains the data on it.

.MYI Each *MyISAM* table has `.MYI` (MYIndex) file which contains the table's indexes.

.exp Files with the `.exp` extension are created by *Percona XtraBackup* per each *InnoDB* tablespace when the `xtrabackup --export` option is used on prepare. These files can be used to import those tablespaces on *Percona Server for MySQL* 5.5 or lower versions, see *restoring individual tables*".

.MRG Each table using the **`MERGE`** storage engine, besides of a *.frm* file, will have *.MRG* file containing the names of the *MyISAM* tables associated with it.

.TRG File containing the Triggers associated to a table, e.g. *:file:'mytable.TRG*. With the *.TRN* file, they represent all the Trigger definitions.

.TRN File containing the Triggers' Names associated to a table, e.g. *:file:'mytable.TRN*. With the *.TRG* file, they represent all the Trigger definitions.

.ARM Each table with the **`Archive Storage Engine`** has `.ARM` file which contains the metadata of it.

.ARZ Each table with the **`Archive Storage Engine`** has `.ARZ` file which contains the data of it.

.CSM Each table with the **`CSV Storage Engine`** has `.CSM` file which contains the metadata of it.

.CSV Each table with the **`CSV Storage`** engine has `.CSV` file which contains the data of it (which is a standard Comma Separated Value file).

.opt *MySQL* stores options of a database (like charset) in a file with a `.opt` extension in the database directory.

.par Each partitioned table has .par file which contains metadata about the partitions.

CHAPTER

EIGHTEEN

INDEX OF FILES CREATED BY PERCONA XTRABACKUP

- Information related to the backup and the server
  - **backup-my.cnf** This file contains information to start the mini instance of InnoDB during the *xtrabackup --prepare*. This is **NOT** a backup of original `my.cnf`. The InnoDB configuration is read from the file `backup-my.cnf` created by **innobackupex** when the backup was made. *xtrabackup --prepare* uses InnoDB configuration from `backup-my.cnf` by default, or from *xtrabackup --defaults-file*, if specified. InnoDB configuration in this context means server variables that affect data format, i.e. `innodb_page_size` option, `innodb_log_block_size`, etc. Location-related variables, like `innodb_log_group_home_dir` or `innodb_data_file_path` are always ignored by *xtrabackup --prepare*, so preparing a backup always works with data files from the backup directory, rather than any external ones.
  - **xtrabackup_checkpoints** The type of the backup (e.g. full or incremental), its state (e.g. prepared) and the *LSN* range contained in it. This information is used for incremental backups. Example of the `xtrabackup_checkpoints` after taking a full backup:

    ```
    backup_type = full-backuped
    from_lsn = 0
    to_lsn = 15188961605
    last_lsn = 15188961605
    ```

    Example of the `xtrabackup_checkpoints` after taking an incremental backup:

    ```
    backup_type = incremental
    from_lsn = 15188961605
    to_lsn = 15189350111
    last_lsn = 15189350111
    ```

  - **xtrabackup_binlog_info** The binary log file used by the server and its position at the moment of the backup. Result of the **SHOW MASTER STATUS**.
  - **xtrabackup_binlog_pos_innodb** The binary log file and its current position for *InnoDB* or *XtraDB* tables.
  - **xtrabackup_binary** The **xtrabackup** binary used in the process.
  - **xtrabackup_logfile** Contains data needed for running the: *xtrabackup --prepare*. The bigger this file is the *xtrabackup --prepare* process will take longer to finish.
  - **<table_name>.delta.meta** This file is going to be created when performing the incremental backup. It contains the per-table delta metadata: page size, size of compressed page (if the value is 0 it means the tablespace isn't compressed) and space id. Example of this file could looks like this:

```
page_size = 16384
zip_size = 0
space_id = 0
```

- Information related to the replication environment (if using the *xtrabackup --slave-info* option):
  - **xtrabackup_slave_info** The CHANGE MASTER statement needed for setting up a slave.
- Information related to the *Galera* and *Percona XtraDB Cluster* (if using the *xtrabackup --galera-info* option):
  - **xtrabackup_galera_info** Contains the values of wsrep_local_state_uuid and wsrep_last_committed status variables

CHAPTER

NINETEEN

TRADEMARK POLICY

This Trademark Policy is to ensure that users of Percona-branded products or services know that what they receive has really been developed, approved, tested and maintained by Percona. Trademarks help to prevent confusion in the marketplace, by distinguishing one company's or person's products and services from another's.

Percona owns a number of marks, including but not limited to Percona, XtraDB, Percona XtraDB, XtraBackup, Percona XtraBackup, Percona Server, and Percona Live, plus the distinctive visual icons and logos associated with these marks. Both the unregistered and registered marks of Percona are protected.

Use of any Percona trademark in the name, URL, or other identifying characteristic of any product, service, website, or other use is not permitted without Percona's written permission with the following three limited exceptions.

First, you may use the appropriate Percona mark when making a nominative fair use reference to a bona fide Percona product.

Second, when Percona has released a product under a version of the GNU General Public License ("GPL"), you may use the appropriate Percona mark when distributing a verbatim copy of that product in accordance with the terms and conditions of the GPL.

Third, you may use the appropriate Percona mark to refer to a distribution of GPL-released Percona software that has been modified with minor changes for the sole purpose of allowing the software to operate on an operating system or hardware platform for which Percona has not yet released the software, provided that those third party changes do not affect the behavior, functionality, features, design or performance of the software. Users who acquire this Percona-branded software receive substantially exact implementations of the Percona software.

Percona reserves the right to revoke this authorization at any time in its sole discretion. For example, if Percona believes that your modification is beyond the scope of the limited license granted in this Policy or that your use of the Percona mark is detrimental to Percona, Percona will revoke this authorization. Upon revocation, you must immediately cease using the applicable Percona mark. If you do not immediately cease using the Percona mark upon revocation, Percona may take action to protect its rights and interests in the Percona mark. Percona does not grant any license to use any Percona mark for any other modified versions of Percona software; such use will require our prior written permission.

Neither trademark law nor any of the exceptions set forth in this Trademark Policy permit you to truncate, modify or otherwise use any Percona mark as part of your own brand. For example, if XYZ creates a modified version of the Percona Server, XYZ may not brand that modification as "XYZ Percona Server" or "Percona XYZ Server", even if that modification otherwise complies with the third exception noted above.

In all cases, you must comply with applicable law, the underlying license, and this Trademark Policy, as amended from time to time. For instance, any mention of Percona trademarks should include the full trademarked name, with proper spelling and capitalization, along with attribution of ownership to Percona LLC and/or its affiliates. For example, the full proper name for XtraBackup is Percona XtraBackup. However, it is acceptable to omit the word "Percona" for brevity on the second and subsequent uses, where such omission does not cause confusion.

In the event of doubt as to any of the conditions or exceptions outlined in this Trademark Policy, please contact trademarks@percona.com for assistance and we will do our very best to be helpful.

CHAPTER

TWENTY

VERSION CHECKING

Some Percona software contains "version checking" functionality which is a feature that enables Percona software users to be notified of available software updates to improve your environment security and performance. Alongside this, the version check functionality also provides Percona with information relating to which software versions you are running, coupled with the environment confirmation which the software is running within. This helps enable Percona to focus our development effort accordingly based on trends within our customer community.

The purpose of this document is to articulate the information that is collected, as well as to provide guidance on how to disable this functionality if desired.

Usage

Version Check was implemented in Percona Toolkit 2.1.4, and was enabled by default in version 2.2.1. Currently it is supported as a `--[no]version-check` option by a number of tools in Percona Toolkit, Percona XtraBackup, and PMM (Percona Monitoring and Management).

When launched with Version Check enabled, the tool that supports this feature connects to a Percona's *version check service* via a secure HTTPS channel. It compares the locally installed version for possible updates, and also checks versions of the following software:

- Operating System
- Percona Monitoring and Management (PMM)
- MySQL
- Perl
- MySQL driver for Perl (DBD::mysql)
- Percona Toolkit

Then it checks for and warns about versions with known problems if they are identified as running in the environment.

Each version check request is logged by the server. Stored information consists of the checked system unique ID followed by the software name and version. The ID is generated either at installation or when the *version checking* query is submitted for the first time.

Note: Prior to version 3.0.7 of Percona Toolkit, the system ID was calculated as an MD5 hash of a hostname, and starting from Percona Toolkit 3.0.7 it is generated as an MD5 hash of a random number. Percona XtraBackup continues to use hostname-based MD5 hash.

As a result, the content of the sent query is as follows:

```
85624f3fb5d2af8816178ea1493ed41a;DBD::mysql;4.044
c2b6d625ef3409164cbf8af4985c48d3;MySQL;MySQL Community Server (GPL) 5.7.22-log
85624f3fb5d2af8816178ea1493ed41a;OS;Manjaro Linux
85624f3fb5d2af8816178ea1493ed41a;Percona::Toolkit;3.0.11-dev
85624f3fb5d2af8816178ea1493ed41a;Perl;5.26.2
```

Disabling Version Check

Although the *version checking* feature does not collect any personal information, you might prefer to disable this feature, either one time or permanently. To disable it one time, use `--no-version-check` option when invoking the tool from a Percona product which supports it. Here is a simple example which shows running pt-diskstats tool from the Percona Toolkit with *version checking* turned off:

```
pt-diskstats --no-version-check
```

Disabling *version checking* permanently can be done by placing `no-version-check` option into the configuration file of a Percona product (see correspondent documentation for exact file name and syntax). For example, in case of Percona Toolkit this can be done in a global configuration file `/etc/percona-toolkit/percona-toolkit.conf`:

```
# Disable Version Check for all tools:
no-version-check
```

In case of Percona XtraBackup this can be done in its configuration file in a similar way:

```
[xtrabackup]
no-version-check
```

Frequently Asked Questions

Why is this functionality enabled by default?

We believe having this functionality enabled improves security and performance of environments running Percona Software and it is good choice for majority of the users.

Why not rely on Operating System's built in functionality for software updates?

In many environments the Operating Systems repositories may not carry the latest version of software and newer versions of software often installed manually, so not being covered by operating system wide check for updates.

Why do you send more information than just the version of software being run as a part of version check service?

Compatibility problems can be caused by versions of various components in the environment, for example problematic versions of Perl, DBD or MySQL could cause operational problems with Percona Toolkit.

Part IX

Indices and tables

- genindex
- search

INDEX

Symbols

C

D

I