mysqldump for Developers

Peter Lavin

2012-Dec-16

Table of Contents

Once you have created a database on your development server you will eventually need to copy it to a production environment. The best way to do this is to use the **mysqldump** program, one of the numerous programs that are part of the MySQL package. As a developer, you may find that this client program is second only to **mysql** in usefulness. This command can copy all of your databases, a specific database or just one table within a database. You can even select specific records within a specific table, copy only the structure of a database or copy both the structure and the data.

You also have a choice of how the data is formatted. Typically **mysqldump** is used to export an SQL script of a database but you can export comma-separated text files for use in a spreadsheet or you can create XML files; all this simply by specifying a different command line option.

Database development work is usually done on a local machine and when complete, the database is transferred to a production server. Since **mysqldump** creates a plain text file, in virtually every case, the operating system (OS) of your development environment is irrelevant to the OS used in the production environment. You can create your database locally on a Mac or Windows, dump it, and import it into a MySQL server running under Linux, making no changes whatsoever. Likewise, you can develop under Linux and deploy to Windows.

Usually, the results of executing **mysqldump** are redirected to a file using the redirection operator, '>'. The following example dumps a database to a file named `output.sql`:

```
shell> mysqldump db_name > output.sql
```

By default output is sent to the screen, but usually this is not very helpful. If you only want to inspect a limited selection of data or view the structure of a table, for example, it is easier to use **mysql** with the `--execute` option as shown in the following:

```
shell> mysql db_name -e "SHOW CREATE TABLE table_name\G"
```

Transferring a database from one machine to another is best accomplished by importing an SQL file created by redirecting the output of **mysqldump**. This article deals with the various ways that databases (or parts of databases) can be exported using this utility. However, for the sake of simplicity, most of the examples won't redirected output to a file.

1. Options You Must Know

The **mysqldump** command has numerous options. They are all useful, in specific circumstances, but many of them are used infrequently by the average developer. This article deals in detail with those that are commonly used and those that aren't used as frequently, but are regarded as indispensable. When appropriate, closely related options are also explained. For example, you can't really talk about the `--tables` option without mentioning the `--databases` option.

For options that are not described here use the `--help` option and refer to the documentation on the MySQL website. Specifying an option at the command line can sometimes save you hours of work. For example, imagine writing a program to convert all database objects into XML format and compare that to specifying the `--xml` option at the command line.

As with most client programs, the **mysqldump** command requires the `--host`, `--user` and `--password` options and, depending upon your OS, the `--port` option. The `--host` and `--port` options default to `localhost` and `3306` respectively and if you set up a configuration file, you need not worry about specifying the `--user` and `--password` options from the command line; these options should be set in the `[client]` section of your configuration file.

The most commonly used options, showing the long form followed by the short form (if available), are as follows:

- `--all-databases` (`-A`) – Backup all databases.
- `--databases` *`db_one db_two ...`* (`-B`) – Dump the specified databases.
- `--no-create-db` (`-n`) – Do not generate `CREATE DATABASE` statement(s).
- `--no-create-info` (`-t`) – Do not generate `CREATE TABLE` statement(s).
- `--no-data` (`-d`) – Do not write any data, copy the structure only.
- `--opt` – This option is a group of options and it is on by default.
- `--where='`*`where_condition`*`'` (`-w '`*`where_condition`*`'`) – Export only the records that match the `WHERE` clause.
- `--xml` (`-X`) – Create output in XML format.

These options are dealt with in detail in the following sections. Not every option warrants its own section and where it makes sense related options are discussed together.

1.1. Dumping a Single Database

In many circumstances you will want to dump the entire contents of a single database. To do this no options need be specified. The **mysqldump** command treats any text not preceded by a single or double dash as a database name. To dump a single database enter **`mysqldump`** ***`db_name`***.

If any text follows the database name and is not preceded by a single or double dash it is assumed to be a table name and not an option. Dumping a specific table or number of tables is done in the following way:

```
shell> mysqldump db_name table_one table_two
```

In this case the first word following **mysqldump** is treated as a database name and any other text is treated as a table name; `table_one` and `table_two` from the database `db_name` are the only tables included in the database dump.

You can of course specify other options along with a database and tables. Options can even be specified between the database name and the table name(s). The following command executes without any problems:

```
shell> mysqldump db_name table_one table_two  --xml
```

1.2. The `opt` Options

The `--opt` option is a group of options that are turned on by default. You need to know about them because there are circumstances where you will want to turn some of these options off. This group of options is made up of the following individual options:

- `--add-drop-table` – Add a `DROP TABLE` statement before the `CREATE TABLE` statement to the dump file.
- `--add-locks` – Add `LOCK TABLES` and `UNLOCK TABLES` statements to the dump file.
- `--create-options` – Include all the MySQL-specific table creation options.
- `--disable-keys` – Disabling keys speeds up `INSERT` statements.
- `--extended-insert` – Instead of creating a single `INSERT` statements for all records, create one statement for each record in a table.
- `--lock-tables` – During the mysqldump process, lock tables.
- `--quick` – Use this option to conserve memory during the mysqldump process.
- `--set-charset` – Add a `SET NAMES` *`default_char_set`* to the output file.

The performance related options such as `--disable-keys` may be of concern if you are uploading a large database to a busy server but the options that will most concern developers are `--add-drop-table`, `--create-options` and `--extended-insert`. There are circumstances where you may not want to drop tables; when you are adding data to an existing database, for example. If you want to export to a database other than MySQL, you may want to turn `--create-options` off since they are MySQL-specific. The `--extended-insert` option creates a separate `INSERT` statement for every record so using this option may make it easier to edit the dump file. Additionally, you may wish to transfer your data to a different relational database management system (RDBMS) that doesn't support the extended INSERT syntax.

You can turn off the `--opt` group of options using the `--skip-opt` option. If you wish to use some of the `--opt` options and not others, use `--skip-opt` followed by the option or options you wish to turn on. For example, to turn off all of the `--opt` group except the `--add-drop-table` option use mysqldump in the following way:

```
shell> mysqldump db_name --skip-opt --add-drop-table
```

When using the `--skip-opt` option, the order of the options is important. The `--add-drop-table` option must follow the `--skip-opt` option otherwise the `--skip-opt` option will turn the `--add-drop-table` option off.

1.3. The `all-databases` Option

This is a very useful option for copying all databases, all database objects and all the data on a specific server. It is the easiest way of backing up the entire contents of a server, though it need not always be used this way; it can be used with other options. For example, if used with the `--no-create-db` option then `CREATE DATABASE` statements will not be generated.

If you want to move everything on one server over to another server then this is the command to use. All you need do is specify the **mysqldump** command followed by the `--all-databases` option.

```
shell> mysqldump --all-databases
```

One thing that you *must* know when using only this option is that the SQL script file created by **mysqldump** drops each table before recreating it. (This is because `--add-drop-table` is on by default as shown in Section 1.2, “The `opt` Options”[2].) If you are creating a backup of an existing server then you need not be concerned. However, if the target server has databases that you wish to preserve intact, then you need to exercise a bit of care when importing a file created using this option. If you have databases on the target server with the same name as databases on the source server, then you run the risk of overwriting existing tables. There is no `--dry-run` option to test the consequences of running a script file, so back up the target server beforehand if you have any concerns. You can also use the `--no-create-info` option so that tables common to the same databases will be not be dropped, but again caution should be exercised. You run the risk of possibly corrupting or overwriting existing data.

Another consideration when you create a dump of all databases, is that you are creating a copy of the `mysql` database, the built-in database that keeps track of user passwords and privileges. When you import this dump into a different server you run the risk of overwriting the existing tables in the `mysql` database. *Be sure that this is what you want to do.*

*While the **mysqldump** command does copy the `mysql` database it ignores the `INFORMATION_SCHEMA` database.*

It would be nice if there was an `--ignore-database` *`db_name`* option that you could use along with the `--all-databases` option so that you could dump all databases except those specified, but no such option exists. There is no easy way to ignore a specific database. As an alternative you can use the `--databases` (see next section for more detail) option and name all the databases you wish to copy. If this is too tedious, another approach is to dump all databases and then manually remove from the output file any SQL statements related to unwanted databases—quite feasible if you are not dumping any data.

1.4. The `databases` Option

When you want to dump more than one database but not all the databases, use the `--databases` option. This option is used in the following way:

```
shell> mysqldump --databases db_one db_two ...
```

The same warning that applied to the `--all-databases` option applies here; this option used by itself creates a script that drops and recreates existing tables on the target server so once the file is executed any existing data will be lost. If you are creating new databases on the destination server then this is not an issue. On the other hand, if all you want to do is transfer data into existing tables, use the `--no-create-info` option as well as the `--databases` option. Use the `--no-create-info` option and no database objects will be created.

When you import data into a table that already has records in it, there is always a chance that there will be duplicate data and perhaps duplicate primary keys. What happens when this occurs is determined by the server option, `sql_mode`. How this option is set depends upon circumstances.

If the databases you are dumping have only test data in them and you want the database objects only, then use the `--no-data` option.

1.4.1. The `tables` Option

The `--tables` option is mentioned here not because it is particularly useful but because it can be misleading. The terse description of this option given on the MySQL website, "Override the `--databases` option", does not really explain how it is used and can give the impression that you can dump a number of databases and, at the same time, select tables from one or more of those databases.

In Section 1.1, “Dumping a Single Database”[2] you saw how **mysqldump** treats text not explicitly preceded by an option; namely the first word is treated as a database name and any other word or words as table names—effectively the `--tables` option is on by default. The `--databases` option turns off this default behavior and treats all text as database names. The `--tables` option exists solely to revert to the default and turn the `--databases` option off.

Find below a syntactically correct but unnecessary use of the `--tables` option:

```
shell> mysqldump -B db_one --tables table_one table_two
```

Issue the command above and `table_one` and `table_two` will be dumped from the `db_one` database because `-B` turns on the databases option and the `--tables` option negates it. *You'll get exactly the same result using only the database name and the table names.*

From the command line it doesn't make sense to turn on the `--databases` option and then immediately turn it off using the `--tables` option. You can simply execute the statement **`mysqldump`** ***`db_one table_one table_two`***. In what circumstances is the `--tables` option useful then? Using the `--tables` option only makes sense when you are overriding a `--databases` option set from a configuration file and this is the sole reason for the existence of the `--tables` option.

The preceding code example also indicates the one circumstance where order of options is important: If one option negates another, then the last option specified is the one in effect. Otherwise, the order of options is irrelevant.

The `--tables` option is similar to the MySQL client program option `--no-tee`—it also doesn't make sense unless the `--tee` option, has been set in a configuration file. The default action of the client program is to send output to the screen and the `--tee` option redirects output to a file. For this reason the `--no-tee` option only makes sense from the command line when `--tee` has already been set in a configuration file.

One lesson to be learned from this is that options specified at the command line have the highest precedence, overriding configuration file settings.

The `--ignore-table` option is much more likely to be useful than is the `--tables` option. This option lets you ignore specific tables when dumping a database. For example, if you wish to dump nineteen of twenty tables in a database it is much easier to specify the one you wish to ignore rather than the nineteen that you are interested in.

1.5. The `where` Option

The `--where` option functions in much the same way that an SQL `WHERE` clause does, though it is more limited. This option is used in the following way:

```
shell> mysqldump db_name table_name -w 'field_name=value'
```

The use of quotation marks with the `--where` option is only required if there is white space in the `WHERE` clause.

This option is only useful if you are dumping the data from a table and are interested in a portion of that data. Since this option limits you to referencing only one table and cannot, for example, be used with a `JOIN` clause, the data that you can extract is somewhat limited. Fortunately, **mysqldump** is not the only way to export data to file. There are other methods of dumping data from a server typically using the MySQL client program.

1.6. The `--xml` Option

The `--xml` option doesn't really require a lot of explanation–it is used for creating an XML file. However, it does illustrate the point mentioned earlier; some options can save you a lot of work. Find below an example of what this option can do. What follows is the result of executing **`mysqldump sakila city -X`** (The sakila database is a demonstration database available for download from the MySQL site):

```
<?xml version="1.0"?>
<mysqldump xmlns:xsi="http://www.w3.org/2001/XMLSchema-instance">
  <database name="sakila">
    <table_structure name="city">
      <field Field="city_id" Type="smallint(5) unsigned" Null="NO"
      Key="PRI" Extra="auto_increment" />
      <field Field="city" Type="varchar(50)" Null="NO" Key=""
      Extra="" />
      <field Field="country_id" Type="smallint(5) unsigned"
      Null="NO" Key="MUL" Extra="" />
      <field Field="last_update" Type="timestamp" Null="NO"
      Key="" Default="CURRENT_TIMESTAMP" Extra="" />
      <key Table="city" Non_unique="0" Key_name="PRIMARY"
      Seq_in_index="1"
          Column_name="city_id" Collation="A" Cardinality="427"
          Null="" Index_type="BTREE" Comment="" />
      <key Table="city" Non_unique="1" Key_name="idx_fk_country_id"
      Seq_in_index="1"
          Column_name="country_id" Collation="A" Cardinality="427"
          Null="" Index_type="BTREE" Comment="" />
      <options Name="city" Engine="InnoDB" Version="10"
      Row_format="Compact"
          Rows="427"  Avg_row_length="115" Data_length="49152"
          Max_data_length="0"  Index_length="16384" Data_free="0"
          Auto_increment="601"  Create_time="2008-03-15 14:38:01"
          Collation="utf8_general_ci" Create_options=""
          Comment="InnoDB free: 4096 kB; (`country_id`)
          REFER `sakila/country`(`country_id`) ON UPD" />
    </table_structure>
    <table_data name="city">
      <row>
        <field name="city_id">1</field>
        <field name="city">A Corua (La Corua)</field>
        <field name="country_id">87</field>
        <field name="last_update">2006-02-15 09:45:25</field>
      </row>
      ...
   </table_data>
  </database>
</mysqldump>
```

For the sake of brevity this example shows only one record from the `city` table but the complete structure and contents are dumped. You can apply other options as you see fit. What you cannot change is the format of the output.

There is no **mysqldump** option to dump in HTML format. However, you can use the `--html` option with the MySQL client program, **mysql**, and redirect output to file. To dump both the `CREATE` statements for database objects and to dump the data as well, you will need to execute a number of SQL statements.

2. A Few Use Cases

As noted at the beginning of this article, the **mysqldump** command is most commonly used to back up the contents of a database server, migrate to another server or to dump a specific database. We have already seen how these tasks are accomplished. This section examines how some of the other options described here might be used.

2.1. Dumping Data Only

Suppose you have data in a MySQL table that you wish to copy into an existing table in a different RDBMS that does not support MySQL's extended INSERT syntax. Assuming that the table structures in the two different databases are identical, the appropriate options might be as follows:

```
shell> mysqldump db_name table_name --skip-opt --no-create-info > output.sql
```

Use the `--skip-opt` option to turn off `--extended-insert`; doing this ensures that individual `INSERT` statements are created for all the records in the specified table. The other options turned off by `--skip-opt` don't have any bearing in this situation so we can safely ignore them. Since we are only interested in the data we also want to suppress the DDL statements, hence the `--no-create-info` option.

Sometimes it is more useful to dump data into a text file of comma or tab separated values than to create a dump file of `INSERT` statements. To do this use a `SELECT ... INTO OUTFILE` statement.

2.2. Dumping the Database Structure Only

If your database contains test data then prior to copying the database to your production server, you may well want to dump the database structure, ignoring the data. This is easily done in the following way:

```
shell> mysqldump db_name --no-data > output.sql
```

Equally, you could remove all data prior to dumping the database, but in some circumstances you may wish to preserve data on your development server.

2.3. Comparing Table Data

Turning off the `--extended-insert` option is also useful if you want to compare data. Suppose you have the same table structure on different servers and you want to know how the data differs. You can dump the data from both tables and use a `diff` utility to determine the differences. Data comparison is much simpler if you turn off `--extended-insert`, creating a separate `INSERT` statement for each record. To ensure that both files are in the same order use the `--order-by-primary` option. This is also one of the rare occasions where you might also use the `--skip-dump-date` and `--skip-comments` options, thereby suppressing two differences between the dump files. (Unless you dump the files simultaneously, the dump dates will differ and the `--comments` option writes the host name to the dump file—given the current scenario, the host values will necessarily differ.)

```
shell> mysqldump db_name table_name --skip-opt \
  --order-by-primary --skip-dump-date --skip-comments > local.sql
shell> mysqldump db_name table_name -h remote_host --skip-opt \
  --order-by-primary --skip-dump-date --skip-comments > remote.sql
```

Since we are only interested in the data we could also turn off the creation of DDL statements by using the `--no-create-info` option but it is not strictly necessary and does make for an even more unwieldy command-line command.

2.3.1. Frequently Specified Options

If you do find that you are constantly using the same options and entering them at the command line becomes tedious, you can add them to your configuration file in the `mysqldump` section. When changing the options of a program such as **mysqldump** there is no need to restart the server—just change the configuration file. For example, if you are constantly skipping the `opt` group of options adjust your configuration file to read:

```
[mysqldump]
skip-opt
```

If and when you do need the `opt` group of options, switch them on by specifying `--opt` from the command line. The `--opt` option specified at the command line will negate the `--skip-opt` option in your configuration file because options specified at the command line take precedence over options specified in a configuration file.

*The **mysqldump** command has exactly the same group of options related to configuration files as does the MySQL client. There is an option that displays the options that will be read from file and another option to override the default configuration file with another configuration file.*

2.4. Ignoring Specific Tables

If you are converting data from a flat table database such as a spreadsheet, your development database may contain interim tables that you do not want to dump. The easiest solution would be to drop the interim tables from the database but there are circumstances where you may want to preserve such tables. For this situation use the following syntax:

```
shell> mysqldump db_name --ignore-table=db_name.interim_one \
  --ignore-table=db_name.interim_two > dump.sql
```

Each table that is being ignored must be specified separately and the database name and the table name must be specified.

*Depending upon circumstances, it may be easier to dump data using the MySQL client program rather than by using **mysqldump**.*

2.5. Standard SQL

In some circumstances you may want to remove MySQL-specific SQL. In this case you need to turn off the `--create-options` option because it includes the MySQL-specific table creation options. This option is one of the `opt` group of options so, again, the `--skip-opt` option is all that is needed.

```
shell> mysqldump db_name table_name --skip-opt > sqlite.sql
```

With `--create-options` turned off, non-standard keywords such as `AUTO_INCREMENT` and `ENGINE` are removed from the `CREATE TABLE` statement; if they are left in, table creation will fail for most other RDBMSes. However, turning `--create-options` off does not mean that the DDL statement will necessarily succeed. For example, if the original MySQL table contains an `enum` field, table creation will also fail.

The option `--skip-opt` also turns off `--extended-insert` creating separate `INSERT` statements for each record thereby making it possible to import the data into a SQLite database, for example. Depending upon the nature of your data and the table structure, migrating to a SQLite database *may* be as simple as taking the dump file 'as is' and issuing the following command:

```
shell> sqlite3 new_db < sqlite.sql
```

However, you may have to remove any `SET` statements that the dump file contains and change some data types. For example, any fields that are `int` will have to become `integer`.

If you only want to export data, the problem becomes much simpler.

The ability to export data and DDL statements in standard SQL can be very helpful if you maintain data in a MySQL database and regularly export it. For example, if you have an Android application that uses data from a MySQL database you must export it in a format compatible with SQLite. Such an export file can be used as a resource file in your Android application making it relatively easy to keep a MySQL and a SQLite database in sync.

If you are migrating to another RDBMS, then you should also examine the `--compatible` option. If you are migrating to PostgreSQL for example, you can issue the command **`mysqldump`** ***`db_name`*** **`--compatible=postgresql`**. This option doesn't guarantee full compatibility but will make the output more compatible with the target RDBMS.

3. Uploading To a Production Server

Earlier we noted that the file created by **mysqldump** is compatible across different OSes because it is plain text. Even the different line separators used by the different OSes don't usually cause any problems. However, you may run into version compatibility issues if you use one version of MySQL for your development database and a different one as your production server. For example, support for triggers was only introduced beginning with MySQL 5.0 so this type of database object cannot be used in MySQL version 4.1. However, if you take a little care in the design phase, you can easily avoid this type of issue.

If you have remote access to your production server then you may be able to upload your file from the command line on your local machine in the following fashion:

```
shell> mysql -h production_server.com -u user_name -p  < dump.sql
```

Here the redirection option is used to copy the dump file to your production server. Unlike other examples given here, the example above shows the host, user name and password options. This is done to emphasize that these values are likely not the values contained in your local configuration file (and even if they were ...). If you have set up your `my.ini` or `my.cnf` file, then the values in that file are assumed to apply to the local development server not a remote production server. Your user name and password may be the same, but the host will certainly differ—by definition it cannot be `localhost`.

That said, it is quite likely that you will not be able to connect to an online server in the above fashion. For security reasons most servers are configured to accept connections only from `localhost` so uploading a dump file from a remote location may not be possible. Don't lose heart though—in this case you can copy your data file to the production server and then log in to the server hosting MySQL using **ssh**. Having logged in to the host server, you can then connect to the MySQL server locally and import the file from the command line.

3.1. Note About the Output of mysqldump

The file created by **mysqldump** is a MySQL script file and this file is, for the most part, readily intelligible. Typically a dump file recreates the database structure and populates the tables with data. However, the meaning of lines such as the following is not immediately evident.

```
/*!40000 ALTER TABLE `table_name` DISABLE KEYS */;
```

The line above acts like a comment depending upon the version number of MySQL. It will execute if the dump file is imported into a version of MySQL later than version 4.0, otherwise it will be ignored. This is very helpful in assuring that the syntax of the dump file will be compatible across most versions of MySQL.

Additionally, this commenting style is typically ignored by other database servers so it should not interfere if you plan to use a dump file with a RDBMS other than MySQL.

These conditional comments do not guarantee that any file created by mysqldump will successfully import into any version of MySQL. For example, if you create database objects, such as views, that are not supported by an earlier version of MySQL, importation will fail.

3.2. Using phpMyAdmin

The more common scenario for uploading a dump file is to use a MySQL database management program such as phpMyAdmin. Files created by **mysqldump** can be imported using phpMyAdmin. The only obstacle that you may encounter is a limitation on the size of the dump file. HTTP servers usually impose limits on the size of files that can be uploaded and, since phpMyAdmin runs in a web browser, such limits would apply. This is often only a consideration when your database contains binary data and in such cases you can dump different tables to different dump files creating a series of smaller files. If files are still too large, you can break down table data files into smaller units using the `--where` option.

Another solution to this problem, is to copy the dump file to the server using an ftp program or a secure copy utility. Once the file is copied to the server you will need shell access to perform the installation from the command line as shown in Section 3, “Uploading To a Production Server” [8].

4. The Limits of mysqldump

You have seen a number of examples of the kinds of files you might create using **mysqldump** and you've seen examples of how options might be combined but in no way is this a definitive treatment. As stated earlier, I would encourage you to examine all the features by using the `--help` option and by visiting the MySQL documentation website. You may find the exact option that suits your current needs.

The **mysqldump** command is especially useful for backing up data and copying the structure and contents of a database. It is also helpful for data transfer and to some extent for migrating to a different RDBMS. However, we have already seen that depending upon circumstances, the MySQL client might be a better way of dumping or viewing data.

If you wish to dump data derived from more than one table you might find it easier to use a `SELECT INTO OUTFILE` statement rather than **mysqldump**. Dumping data using the client program gives you the freedom to use joins and to select from more than one table. This is especially important if you plan to export to a flat file such as a spreadsheet. Alternately, you can create a view of the data you are interested in and use that view with a `SELECT INTO OUTFILE` statement.

When exporting data to spreadsheets you need to be able to create comma separated values (CSV) files. You can create a CSV file by using the `--fields-terminated-by` option with mysqldump. This option and related options such as `--lines-terminated-by` have the same meaning as they do in `LOAD DATA INFILE` statements and, while they can be used with **mysqldump**, they are better exploited using `SELECT INTO OUTFILE` statements.

5. About the Author

Peter Lavin is a technical writer who has been published in a number of print and online magazines. He is the author of Object Oriented PHP, published by No Starch Press and a contributor to PHP Hacks by O'Reilly Media.