3

Using MySQL Programs

This chapter provides a brief overview of the command-line programs provided by MySQL AB and discusses the general syntax for specifying options when you run these programs. Most programs have options that are specific to their own operation, but the option syntax is similar for all of them. Later chapters provide more detailed descriptions of individual programs, including which options they recognize.

MySQL AB also provides three GUI client programs for use with MySQL Server:

- MySQL Administrator: This tool is used for administering MySQL servers, databases, tables, and user accounts.
- MySQL Query Browser: This graphical tool is provided by MySQL AB for creating, executing, and optimizing queries on MySQL databases.
- MySQL Migration Toolkit: This tool helps you migrate schemas and data from other relational database management systems for use with MySQL.

These GUI programs each have their own manuals that you can access at `http://dev.mysql.com/doc/`.

3.1 Overview of MySQL Programs

MySQL AB provides several types of programs:

- The MySQL server and server startup scripts:
  - `mysqld` is the MySQL server.
  - `mysqld_safe`, `mysql.server`, and `mysqld_multi` are server startup scripts.
  - `mysql_install_db` initializes the data directory and the initial databases.
  - MySQL Instance Manager monitors and manages MySQL Server instances.

  Chapter 4, "Database Administration," discusses these programs further.

- Client programs that access the server:
  - `mysql` is a command-line client for executing SQL statements interactively or in batch mode.
  - `mysqladmin` is an administrative client.
  - `mysqlcheck` performs table maintenance operations.
  - `mysqldump` and `mysqlhotcopy` make database backups.
  - `mysqlimport` imports data files.
  - `mysqlshow` displays information about databases and tables.

  Chapter 7, "Client and Utility Programs," discusses these programs further.
- Utility programs that operate independently of the server:
  - `myisamchk` performs table maintenance operations.
  - `myisampack` produces compressed, read-only tables.
  - `mysqlbinlog` is a tool for processing binary log files.
  - `perror` displays the meaning of error codes.

  Chapter 7, "Client and Utility Programs," discusses these programs further.

Most MySQL distributions include all of these programs, except for those programs that are platform-specific. (For example, the server startup scripts are not used on Windows.) The exception is that RPM distributions are more specialized. There is one RPM for the server, another for client programs, and so forth. If you appear to be missing one or more programs, see Chapter 2, "Installing and Upgrading MySQL," for information on types of distributions and what they contain. It may be that you have a distribution that does not include all programs and you need to install something else.

3.2 Invoking MySQL Programs

To invoke a MySQL program from the command line (that is, from your shell or command prompt), enter the program name followed by any options or other arguments needed to instruct the program what you want it to do. The following commands show some sample program invocations. "`shell>`" represents the prompt for your command interpreter; it is not part of what you type. The particular prompt you see depends on your command interpreter. Typical prompts are `$` for `sh` or `bash`, `%` for `csh` or `tcsh`, and `C:\>` for the Windows `command.com` or `cmd.exe` command interpreters.

```
shell> mysql -u root test
shell> mysqladmin extended-status variables
shell> mysqlshow --help
shell> mysqldump --user=root personnel
```

Arguments that begin with a single or double dash ('-', '--') are option arguments. Options typically specify the type of connection a program should make to the server or affect its operational mode. Option syntax is described in Section 3.3, "Specifying Program Options."

Non-option arguments (arguments with no leading dash) provide additional information to the program. For example, the `mysql` program interprets the first non-option argument as a database name, so the command `mysql -u root test` indicates that you want to use the `test` database.

Later sections that describe individual programs indicate which options a program understands and describe the meaning of any additional non-option arguments.

Some options are common to a number of programs. The most common of these are the `--host` (or `-h`), `--user` (or `-u`), and `--password` (or `-p`) options that specify connection parameters. They indicate the host where the MySQL server is running, and the username and password of your MySQL account. All MySQL client programs understand these options; they allow you to specify which server to connect to and the account to use on that server.

You may find it necessary to invoke MySQL programs using the pathname to the `bin` directory in which they are installed. This is likely to be the case if you get a "program not found" error whenever you attempt to run a MySQL program from any directory other than the `bin` directory. To make it more convenient to use MySQL, you can add the pathname of the `bin` directory to your `PATH` environment variable setting. That enables you to run a program by typing only its name, not its entire pathname. For example, if `mysql` is installed in `/usr/local/mysql/bin`, you'll be able to run it by invoking it as `mysql`; it will not be necessary to invoke it as `/usr/local/mysql/bin/mysql`.

Consult the documentation for your command interpreter for instructions on setting your `PATH` variable. The syntax for setting environment variables is interpreter-specific.

3.3 Specifying Program Options

There are several ways to specify options for MySQL programs:

- List the options on the command line following the program name. This is most common for options that apply to a specific invocation of the program.
- List the options in an option file that the program reads when it starts. This is common for options that you want the program to use each time it runs.
- List the options in environment variables. This method is useful for options that you want to apply each time the program runs. In practice, option files are used more commonly for this purpose. However, Section 4.13.2, "Running Multiple Servers on Unix," discusses one situation in which environment variables can be very helpful. It describes a handy technique that uses such variables to specify the TCP/IP port number and Unix socket file for both the server and client programs.

MySQL programs determine which options are given first by examining environment variables, then by reading option files, and then by checking the command line. If an option is

specified multiple times, the last occurrence takes precedence. This means that environment variables have the lowest precedence and command-line options the highest.

You can take advantage of the way that MySQL programs process options by specifying default values for a program's options in an option file. That enables you to avoid typing them each time you run the program, but also allows you to override the defaults if necessary by using command-line options.

3.3.1 Using Options on the Command Line

Program options specified on the command line follow these rules:

- Options are given after the command name.
- An option argument begins with one dash or two dashes, depending on whether it has a short name or a long name. Many options have both forms. For example, `-?` and `--help` are the short and long forms of the option that instructs a MySQL program to display a help message.
- Option names are case sensitive. `-v` and `-V` are both legal and have different meanings. (They are the corresponding short forms of the `--verbose` and `--version` options.)
- Some options take a value following the option name. For example, `-h localhost` or `--host=localhost` indicate the MySQL server host to a client program. The option value tells the program the name of the host where the MySQL server is running.
- For a long option that takes a value, separate the option name and the value by an '=' sign. For a short option that takes a value, the option value can immediately follow the option letter, or there can be a space between: `-hlocalhost` and `-h localhost` are equivalent. An exception to this rule is the option for specifying your MySQL password. This option can be given in long form as `--password=`*`pass_val`* or as `--password`. In the latter case (with no password value given), the program prompts you for the password. The password option also may be given in short form as `-p`*`pass_val`* or as `-p`. However, for the short form, if the password value is given, it must follow the option letter with *no intervening space*. The reason for this is that if a space follows the option letter, the program has no way to tell whether a following argument is supposed to be the password value or some other kind of argument. Consequently, the following two commands have two completely different meanings:

  ```
  shell> mysql -ptest
  shell> mysql -p test
  ```

  The first command instructs `mysql` to use a password value of `test`, but specifies no default database. The second instructs `mysql` to prompt for the password value and to use `test` as the default database.

Some options control behavior that can be turned on or off. For example, the `mysql` client supports a `--column-names` option that determines whether or not to display a row of column names at the beginning of query results. By default, this option is enabled. However, you

may want to disable it in some instances, such as when sending the output of `mysql` into another program that expects to see only data and not an initial header line.

To disable column names, you can specify the option using any of these forms:

```
--disable-column-names
--skip-column-names
--column-names=0
```

The `--disable` and `--skip` prefixes and the `=0` suffix all have the same effect: They turn the option off.

The "enabled" form of the option may be specified in any of these ways:

```
--column-names
--enable-column-names
--column-names=1
```

If an option is prefixed by `--loose`, a program does not exit with an error if it does not recognize the option, but instead issues only a warning:

```
shell> mysql --loose-no-such-option
mysql: WARNING: unknown option '--no-such-option'
```

The `--loose` prefix can be useful when you run programs from multiple installations of MySQL on the same machine and list options in an option file. An option that may not be recognized by all versions of a program can be given using the `--loose` prefix (or `loose` in an option file). Versions of the program that recognize the option process it normally, and versions that do not recognize it issue a warning and ignore it.

Another option that may occasionally be useful with `mysql` is the `--execute` or `-e` option, which can be used to pass SQL statements to the server. The statements must be enclosed by single or double quotation marks. If you wish to use quoted values within a statement, you should use double quotes for the statement, and single quotes for any quoted values within the statement. When this option is used, `mysql` executes the statements and exits.

For example, you can use the following command to obtain a list of user accounts:

```
shell> mysql -u root -p --execute="SELECT User, Host FROM user" mysql
Enter password: ******
+------+-----------+
| User | Host      |
+------+-----------+
|      | gigan     |
| root | gigan     |
|      | localhost |
| jon  | localhost |
| root | localhost |
+------+-----------+
shell>
```

Note that the long form (`--execute`) is followed by an equal sign (`=`).

In the preceding example, the name of the `mysql` database was passed as a separate argument. However, the same statement could have been executed using this command, which specifies no default database:

```
mysql> mysql -u root -p --execute="SELECT User, Host FROM mysql.user"
```

Multiple SQL statements may be passed on the command line, separated by semicolons:

```
shell> mysql -u root -p -e "SELECT VERSION();SELECT NOW()"
Enter password: ******
+------------+
| VERSION()  |
+------------+
| 5.0.19-log |
+------------+
+---------------------+
| NOW()               |
+---------------------+
| 2006-01-05 21:19:04 |
+---------------------+
```

The `--execute` or `-e` option may also be used to pass commands in an analogous fashion to the `ndb_mgm` management client for MySQL Cluster. See Section 9.3.6, "Safe Shutdown and Restart," for an example.

3.3.2 Using Option Files

Most MySQL programs can read startup options from option files (also sometimes called "configuration files"). Option files provide a convenient way to specify commonly used options so that they need not be entered on the command line each time you run a program.

To determine whether a program reads option files, invoke it with the `--help` option (`--verbose` and `--help` for `mysqld`). If the program reads option files, the help message indicates which files it looks for and which option groups it recognizes.

Note: Option files used with MySQL Cluster programs are covered in Section 9.4, "MySQL Cluster Configuration."

On Windows, MySQL programs read startup options from the following files:

Filename	Purpose
WINDIR\my.ini	Global options
C:\my.cnf	Global options
INSTALLDIR\my.ini	Global options
defaults-extra-file	The file specified with `--defaults-extra-file=`*path*, if any

WINDIR represents the location of your Windows directory. This is commonly C:\WINDOWS or C:\WINNT. You can determine its exact location from the value of the WINDIR environment variable using the following command:

```
C:\> echo %WINDIR%
```

INSTALLDIR represents the installation directory of MySQL. This is typically C:*PROGRAMDIR*\MySQL\MySQL 5.0 Server where *PROGRAMDIR* represents the programs directory (usually Program Files on English-language versions of Windows), when MySQL 5.0 has been installed using the installation and configuration wizards. See Section 2.3.4.14, "The Location of the my.ini File."

On Unix, MySQL programs read startup options from the following files:

Filename	Purpose
/etc/my.cnf	Global options
$MYSQL_HOME/my.cnf	Server-specific options
defaults-extra-file	The file specified with --defaults-extra-file=*path*, if any
~/.my.cnf	User-specific options

MYSQL_HOME is an environment variable containing the path to the directory in which the server-specific my.cnf file resides. (This was *DATADIR* prior to MySQL version 5.0.3.)

If MYSQL_HOME is not set and you start the server using the mysqld_safe program, mysqld_safe attempts to set MYSQL_HOME as follows:

- Let *BASEDIR* and *DATADIR* represent the pathnames of the MySQL base directory and data directory, respectively.
- If there is a my.cnf file in *DATADIR* but not in *BASEDIR*, mysqld_safe sets MYSQL_HOME to *DATADIR*.
- Otherwise, if MYSQL_HOME is not set and there is no my.cnf file in *DATADIR*, mysqld_safe sets MYSQL_HOME to *BASEDIR*.

Typically, *DATADIR* is /usr/local/mysql/data for a binary installation or /usr/local/var for a source installation. Note that this is the data directory location that was specified at configuration time, not the one specified with the --datadir option when mysqld starts. Use of --datadir at runtime has no effect on where the server looks for option files, because it looks for them before processing any options.

MySQL looks for option files in the order just described and reads any that exist. If an option file that you want to use does not exist, create it with a plain text editor.

If multiple instances of a given option are found, the last instance takes precedence. There is one exception: For mysqld, the *first* instance of the --user option is used as a security precaution, to prevent a user specified in an option file from being overridden on the command line.

Note: On Unix platforms, MySQL ignores configuration files that are world-writable. This is intentional, and acts as a security measure.

Any long option that may be given on the command line when running a MySQL program can be given in an option file as well. To get the list of available options for a program, run it with the `--help` option.

The syntax for specifying options in an option file is similar to command-line syntax, except that you omit the leading two dashes. For example, `--quick` or `--host=localhost` on the command line should be specified as `quick` or `host=localhost` in an option file. To specify an option of the form `--loose-`*`opt_name`* in an option file, write it as `loose-`*`opt_name`*.

Empty lines in option files are ignored. Non-empty lines can take any of the following forms:

- *`#comment`*, *`;comment`*

  Comment lines start with '`#`' or '`;`'. A '`#`' comment can start in the middle of a line as well.

- `[`*`group`*`]`

  `group` is the name of the program or group for which you want to set options. After a group line, any option-setting lines apply to the named group until the end of the option file or another group line is given.

- *`opt_name`*

  This is equivalent to `--`*`opt_name`* on the command line.

- *`opt_name=value`*

  This is equivalent to `--`*`opt_name=value`* on the command line. In an option file, you can have spaces around the '`=`' character, something that is not true on the command line. You can enclose the value within single quotes or double quotes, which is useful if the value contains a '`#`' comment character or whitespace.

For options that take a numeric value, the value can be given with a suffix of `K`, `M`, or `G` (either uppercase or lowercase) to indicate a multiplier of 1024, 1024^2 or 1024^3. For example, the following command tells `mysqladmin` to ping the server 1024 times, sleeping 10 seconds between each ping:

```
mysql> mysqladmin --count=1K --sleep=10 ping
```

Leading and trailing blanks are automatically deleted from option names and values. You may use the escape sequences '`\b`', '`\t`', '`\n`', '`\r`', '`\\`', and '`\s`' in option values to represent the backspace, tab, newline, carriage return, backslash, and space characters, respectively.

Because the '`\\`' escape sequence represents a single backslash, you must write each '`\`' as '`\\`'. Alternatively, you can specify the value using '`/`' rather than '`\`' as the pathname separator.

If an option group name is the same as a program name, options in the group apply specifically to that program. For example, the `[mysqld]` and `[mysql]` groups apply to the `mysqld` server and the `mysql` client program, respectively.

The `[client]` option group is read by all client programs (but *not* by `mysqld`). This allows you to specify options that apply to all clients. For example, `[client]` is the perfect group to use to specify the password that you use to connect to the server. (But make sure that the option file is readable and writable only by yourself, so that other people cannot find out your password.) Be sure not to put an option in the `[client]` group unless it is recognized by *all* client programs that you use. Programs that do not understand the option quit after displaying an error message if you try to run them.

Here is a typical global option file:

```
[client]
port=3306
socket=/tmp/mysql.sock

[mysqld]
port=3306
socket=/tmp/mysql.sock
key_buffer_size=16M
max_allowed_packet=8M

[mysqldump]
quick
```

The preceding option file uses *`var_name=value`* syntax for the lines that set the `key_buffer_size` and `max_allowed_packet` variables.

Here is a typical user option file:

```
[client]
# The following password will be sent to all standard MySQL clients
password="my_password"

[mysql]
no-auto-rehash
connect_timeout=2

[mysqlhotcopy]
interactive-timeout
```

If you want to create option groups that should be read by `mysqld` servers from a specific MySQL release series only, you can do this by using groups with names of `[mysqld-4.1]`, `[mysqld-5.0]`, and so forth. The following group indicates that the `--new` option should be used only by MySQL servers with 5.0.x version numbers:

```
[mysqld-5.0]
new
```

Beginning with MySQL 5.0.4, it is possible to use `!include` directives in option files to include other option files and `!includedir` to search specific directories for option files. For example, to include the `/home/mydir/myopt.cnf` file, you can use the following directive:

```
!include /home/me/myopt.cnf
```

To search the `/home/mydir` directory and read option files found there, you would use this directive:

```
!includedir /home/mydir
```

Note: Currently, any files to be found and included using the `!includedir` directive on Unix operating systems *must* have filenames ending in `.cnf`. On Windows, this directive checks for files with the `.ini` or `.cnf` extension.

Note that options read from included files are applied in the context of the current option group. Suppose that you were to write the following lines in `my.cnf`:

```
[mysqld]
!include /home/mydir/myopt.cnf
```

In this case, the `myopt.cnf` file is processed only for the server, and the `!include` directive is ignored by any client applications. However, if you were to use the following lines, the directory `/home/mydir/my-dump-options` is checked for option files by `mysqldump` only, and not by the server or by any other client applications:

```
[mysqldump]
!includedir /home/mydir/my-dump-options
```

If you have a source distribution, you can find sample option files named `my-xxxx.cnf` in the `support-files` directory. If you have a binary distribution, look in the `support-files` directory under your MySQL installation directory. On Windows, the sample option files may be located in the MySQL installation directory (see earlier in this section or Chapter 2, "Installing and Upgrading MySQL," if you do not know where this is). Currently, there are sample option files for small, medium, large, and very large systems. To experiment with one of these files, copy it to `C:\my.cnf` on Windows or to `.my.cnf` in your home directory on Unix.

Note: On Windows, the `.cnf` option file extension might not be displayed.

All MySQL programs that support option files handle the following options. They affect option-file handling, so they must be given on the command line and not in an option file. To work properly, each of these options must immediately follow the command name, with the exception that `--print-defaults` may be used immediately after `--defaults-file` or `--defaults-extra-file`.

- `--no-defaults`

  Don't read any option files.

- `--print-defaults`

  Print the program name and all options that it gets from option files.

- `--defaults-file=`*`file_name`*

  Use only the given option file. *`file_name`* is the full pathname to the file.

- `--defaults-extra-file=`*`file_name`*

  Read this option file after the global option file but (on Unix) before the user option file. *`file_name`* is the full pathname to the file.

In shell scripts, you can use the `my_print_defaults` program to parse option files and see what options would be used by a given program. The following example shows the output that `my_print_defaults` might produce when asked to show the options found in the `[client]` and `[mysql]` groups:

```
shell> my_print_defaults client mysql
--port=3306
--socket=/tmp/mysql.sock
--no-auto-rehash
```

Note for developers: Option file handling is implemented in the C client library simply by processing all options in the appropriate group or groups before any command-line arguments. This works well for programs that use the last instance of an option that is specified multiple times. If you have a C or C++ program that handles multiply specified options this way but that doesn't read option files, you need add only two lines to give it that capability. Check the source code of any of the standard MySQL clients to see how to do this.

Several other language interfaces to MySQL are based on the C client library, and some of them provide a way to access option file contents. These include Perl and Python. For details, see the documentation for your preferred interface.

3.3.3 Using Environment Variables to Specify Options

To specify an option using an environment variable, set the variable using the syntax appropriate for your command processor. For example, on Windows or NetWare, you can set the `USER` variable to specify your MySQL account name. To do so, use this syntax:

```
SET USER=your_name
```

The syntax on Unix depends on your shell. Suppose that you want to specify the TCP/IP port number using the `MYSQL_TCP_PORT` variable. Typical syntax (such as for `sh`, `bash`, `zsh`, and so on) is as follows:

```
MYSQL_TCP_PORT=3306
export MYSQL_TCP_PORT
```

The first command sets the variable, and the `export` command exports the variable to the shell environment so that its value becomes accessible to MySQL and other processes.

For `csh` and `tcsh`, use `setenv` to make the shell variable available to the environment:

```
setenv MYSQL_TCP_PORT 3306
```

The commands to set environment variables can be executed at your command prompt to take effect immediately, but the settings persist only until you log out. To have the settings take effect each time you log in, place the appropriate command or commands in a startup file that your command interpreter reads each time it starts. Typical startup files are `AUTOEXEC.BAT` for Windows, `.bash_profile` for `bash`, or `.tcshrc` for `tcsh`. Consult the documentation for your command interpreter for specific details.

3.3.4 Using Options to Set Program Variables

Many MySQL programs have internal variables that can be set at runtime. Program variables are set the same way as any other long option that takes a value. For example, `mysql` has a `max_allowed_packet` variable that controls the maximum size of its communication buffer. To set the `max_allowed_packet` variable for `mysql` to a value of 16MB, use either of the following commands:

```
shell> mysql --max_allowed_packet=16777216
shell> mysql --max_allowed_packet=16M
```

The first command specifies the value in bytes. The second specifies the value in megabytes. For variables that take a numeric value, the value can be given with a suffix of `K`, `M`, or `G` (either uppercase or lowercase) to indicate a multiplier of 1024, 1024^2 or 1024^3. (For example, when used to set `max_allowed_packet`, the suffixes indicate units of kilobytes, megabytes, or gigabytes.)

In an option file, variable settings are given without the leading dashes:

```
[mysql]
max_allowed_packet=16777216
```

Or:

```
[mysql]
max_allowed_packet=16M
```

If you like, underscores in a variable name can be specified as dashes. The following option groups are equivalent. Both set the size of the server's key buffer to 512MB:

```
[mysqld]
key_buffer_size=512M

[mysqld]
key-buffer-size=512M
```

Note: Before MySQL 4.0.2, the only syntax for setting program variables was `--set-variable=`*`option`*`=`*`value`* (or `set-variable=`*`option`*`=`*`value`* in option files). This syntax still is recognized, but is deprecated as of MySQL 4.0.2.

Many server system variables can also be set at runtime. For details, see Section 4.2.3.2, "Dynamic System Variables."