Installing MySQL

Operating system *Authentication* *Configuration* *Character sets and collations* *MySQL client program*
Installation
*Binary distribution, *Nix* *Source distribution, *Nix* *2nd server, Nix*
Binary distribution, Windows *2nd server, Windows* *Source distribution,Windows*
Setup, testing, other modules
Upgrades *Updating privilege tables* *Time zone support* *Getting started* *New InnoDB engineMySQL*
Connectors *MySQL Workbench* *Notifier* *MySQL Proxy* *MySQL Shell*

A one-step for obtaining MySQL, *PHP*, *Perl*, Apache web server and *PHPMyAdmin* is to download and install *WampServer* or *XAMPP*—everything just works Note that XAMPP installs the MariaDB fork of MySQL. Even easier: install nothing and sign up for *Google Cloud SQL*, implemented with a Google fork of MySQL: backups, upgrades and even replication are looked after, but you'll owe a monthly usage fee.

To install MySQL 5.7 *de novo* with *Connectors*, *WorkBench* and *Notifier* in Windows, the *GUI installer* is useful, combining requirement checks, installation options (development, server only, client only, full or custom) and automatic configuration.

Use this chapter for reference if these conveniences don't work for you. There are *binary* and *source* MySQL distributions. Unless you need to customise MySQL C/C++ code, go with a binary distribution. Installation comes down to five main steps: choose a package for your operating system, download, unpack/install, configure, and test.

What version, what operating system?

Major MySQL versions since 2000 have been, in order: 3.2, 4.0, 4.1, 5.0, 5.1, 6.0, 5.5, 5.6, 5.7, 8.0. Yes 6.0 preceded 5.5; see this *history graphic*. To develop MySQL databases you need to install each major version you use. MySQL production releases are available as free *Community* and as commercial *Enterprise* editions. When upgrading, do *not* skip past major versions.

Versions 3.2, 4.0, and 4.1 are obsolete; 5.0 went into production in 2005, 5.1 in 2008, 5.5 in late 2010, 5.6 in early 2013, 5.7 in late 2015. Version 6.0 appeared in 2007 but disappeared in 2009 when MySQL announced a new release model, and a new 6.0 for 2010. That didn't happen. Instead Brian Aker forked MySQL 6.0 (*www.drizzle.org*) while MySQL 5.5, 5.6 and 5.7 appeared. In September 2016 a development release of 8.0 appeared. So there are six available releases, five of them active:

- ***5.0***: added Stored Routines, Triggers, Views, `information_schema`, XA transactions, NDB CLUSTER and FEDERATED engines. Active 5.0 development ended 31 Dec 2009; 5.0 is available now only as an *archived release*.
- ***5.1***: added *partitions, an event scheduler*, *row-based replication and more logs*, a *plugin API*, NDB CLUSTER storage engine improvements, a *load emulator* and more `information_schema` tables, and the fast, ACID-compliant *PBXT* storage engine plugin. Active development ended 31 Dec 2010.

- ***5.5***: in production since December 2010 and more stable than 5.1, added the *new INNODB engine* with concurrency, thread and lock improvements; *SIGNAL, RESIGNAL*; *LOAD XML and partitioning extensions*; *semisynchronous replication*; enhancements for Solaris; and a *new authentication plugin architecture*.
- ***5.6***: in production since February 2013, adds crash-safe binary logs; PARTITION references in SELECT, INSERT, UPDATE and DELETE commands; UUIDs for each running server instance; delayed replication; row image control in replication; PERFORMANCE_SCHEMA enhancements; better optimisation of JOIN , WHERE and LIMIT; and expanded INFORMATION_SCHEMA INNODB metadata.
- ***5.7***: introduced April 2013, now in production, brings better security, logging and Triggers; subqueries in Views; JSON and derived columns; many bug fixes.
- ***8.0***: a *September 2016* development release, improves InnoDB and moves system tables and the global data dictionary to it; revises partitioning; adds binary strings to bit function scope; improves query parsing and index use; and adds component-based infrastructure, SQL roles, a `utf8mbr` collation. Common Table Expressions have appeared in the 8.0.1 release; persistent global variables are coming.

A *serious password-hacking vulnerability* renders all MySQL 5.0 versions, all 5.1 before 5.1.63, and all 5.5 before 5.5.23 unsafe for production data. A *dangerous exploit vulnerability* was corrected in releases 5.5.52, 5.6.33 and 5.7.15.

An inviting alternative is *MariaDB*: 5.5 is drop-in compatible with MySQL 5.5, but *no MariaDB version* is entirely compatible with MySQL 5.6 or 5.7.

MySQL performance on a given operating system (OS) depends on CPU power, OS kernel and file system quality, thread library stability, kernel capacity for shared multi-processing, thread lock/unlock flexibility, and build stability and robustness. Open source Linux kernels are blazingly fast. Windows programs tend to run 5-7 times faster when ported to Linux. Nine of the world's ten fastest supercomputers run Linux. Other factors being equal, the OS of choice is Linux. Then iwhy isn't MySQL for Linux more popular than for Windows? The richer Windows user interface. It suggests: develop on Windows, deploy on Linux.

Whatever the OS, more memory is better. So are many gigabytes of free disk space. See *Chapter 17* for discussion of hardware configurations.

MySQL installation packages used to change often, less so since 5.1. *NDB CLUSTER* is included in community releases 5.5 and 5.6, but now is *separate*.

MySQL-OS matchings (Table 3-1) are asymmetric, *e.g.,* Windows 10 needs 5.7, which however runs on Windows 7, 8, 10, 2008 Server R2, and Server 2012. For Linux there are also *Yum*, *APT* and *SUSE* repositories. Overall there are four *main package patterns*:

- for Linux: *RedHat Package Manager* (RPM) bundles with server and client, plus bundles with NDB, partition support, client libraries and embedded server;
- for Linux, Solaris, many UNIX flavours, Netware, DEC OSF and MAC OS: packages with the standard server and a server compiled with debugging stubs;
- for many a variety of Linux there is a MySQL installation package tailored specifically for direct commandline or GUI download and installation without browsing the MySQL website, e.g., via `apt-get` in Ubuntu, `YaST` in SuSE, etc;

- for Windows: a full version using the Windows installer, a full version without that installer, and a stripped down "Essentials" version that is best avoided. Since 5.5.11, the embedded server is included only in zips, not in the *.msi* package.

Like package content, installation process details change from release to release. If you stick with the default configuration for a given release, installation is reasonably straightforward. Custom configuration *isn't*, and it plays back into installation, so you must know at least a little about configuration before you start. There are *more than 500* configuration options for MySQL character sets and collations, memory use, file location, transaction systems and behaviour, security, performance, debugging, response to SQL syntax, etc. These options can be set on client or server command line, and/or in text option files. Before installing MySQL, at least skim the next sections on configuration methods, character sets, and the client program *mysql*.

Table 3-1: Operating systems and MySQL

Operating System	Architecture					MySQL Version					
	sparc-32	sparc-64	x86	x86-64	Intel IA64	5.0	5.1	5.5	5.6	5.7	8.0
Debian Linux 4			•	•		•	•				
Debian Linux 5			•	•			•	•			
Debian Linux 6			•	•			•	•	•		
Debian Linux 7, 8			•	•					•	•	
Debian Linux 8			•	•							•
Fedora 24			•	•							•
FreeBSD 10.2											•
Oracle Linux 4			•	•	•	•	•	•			
Oracle Linux 5			•	•		•	•	•	•	•	
Oracle Linux 6			•	•				•	•	•	
OS X 10.5			•	•			•	•			
OS X 10.6			•	•			•	•	•		
OS X 10.7, 10.8				•					•		
OS X 10.9, 10.10				•					•	•	
OS X 10.11				•						•	•
Oracle Linux 7				•				•	•	•	
RHE Linux 3			•	•	•	•	•				
RHE Linux 4			•	•	•	•	•	•			
RHE Linux 5/CentOS 5			•	•		•	•	•	•	•	
RHE Linux 6/CentOS 6			•	•				•	•	•	
RHE Linux 7/CentOS 7				•				•	•	•	•
Solaris 8, 9	•	•	•	•		•	•				
Solaris 10	•					•	•				
Solaris 10 (Update 8+)		•	•	•		•	•	•	•		
Solaris 11		•		•				•	•	•	•
SUSE Ent Linux 9				•	•	•	•				
SUSE Ent Linux 10				•	•	•	•	•			
SUSE Ent Linux 11				•			•	•	•		
SUSE Ent Linux 12				•					•	•	•
Ubuntu 12.04/14.04 LTS			•	•				•	•	•	
Ubuntu 16.04			•	•							•
Windows 2008 Server R2				•			•	•	•	•	
Windows 2012 Server				•				•	•	•	
Windows 7			•	•			•	•	•	•	•
Windows 8			•	•				•	•	•	•
Windows 10			•	•						•	•

Authentication

MySQL has always authenticated users by matching the user connection address (`host`), name (`user`) and `password` to a `mysql.user` table row created directly with CREATE USER or with GRANT (*Chapter 6*). The server skips this when started with `--skip-grant-tables`. MySQL 5.5 and 5.6 added four methods:

1. Since 5.5.7, authentication may use *custom plugins* listed in two new columns of `mysql.user`, `plugin` (the plugin name, default `mysql_native_ password` since 5.7.2) and `authentication_string` to be passed to the plugin. Such a plugin may also return a *proxy username* defined by CREATE USER or GRANT. Validity is per-connection. Further documentation is *here*. See also *Chapter 19* (Configuring databases for security).

2. Since 5.5.10, MySQL includes an `auth_socket` server-side plugin in *plugin_dir* to authenticate clients connecting from localhost through the Unix socket file: with it, only user `'abc'@'localhost' IDENTIFIED WITH auth_socket` can connect through the socket file with the argument `--user='abc'`.

3. Since 5.5.16, MySQL Enterprise ships with server-side Pluggable Authentication Modules (PAM) for Mac OS and Linux and for Windows 2000 or later using native Windows authentication. The Windows plugin uses client identity to distinguish client and group identity, and authenticates with Kerberos if available, otherwise with NTLM.

4. Version 5.6.6 introduced *mysql_config_editor* to write encrypted user, host. password and socket specs for a named *login_path* to an encrypted file. Having stored `login_path_name` specs, invoke the *mysql* client program with

```
mysql --login_path=login_path_name
```

Version 8 has added SQL roles (*Chapter 6*).

Configuration methods

Most MySQL programs, including servers, read configuration variable settings from the command line and from text option files*, my.cnf* in *Nix, *my.ini* in Windows. Many MySQL programs can also be told, on the command line, which option files to read. Documentation for configuration variables is in different parts of the MySQL manual: under *command line options*, under *Database Administration*, under *System Variables*, under *SHOW VARIABLES*, and in the *INNODB section*. Some can be read by running *mysqld --help* or *mysqladmin variables* from a command line.

System variables, their names, and their syntax change often. Thus *Appendix B*, which tabulates variables, their meanings and rules. Traditionally MySQL variable names have been built, like compound words in Finnish, by stringing words together, but (unlike in Finnish) with hyphens between the words, for example `delay-key-write-for-all-tables`. Also traditionally, these MySQL arguments have been given either as command-line arguments, or as lines in an option file. MySQL is now making more configuration variables available to SET/SELECT @@ syntax. But SQL syntax interprets the hyphen as the subtraction operator, so SET/SELECT variables must use underscores rather than hyphens as joiners in their names. MySQL recognises some variable names with dashes *or* underscores, eg `default-week-format`, `default_week_format`.

Using option files

Under Linux, MySQL programs now look for option files in this order:

- `/etc/my.cnf`, for global options,
- `my.cnf` in the MySQL home directory for server-specific options,
- `defaults-extra-file` if specified,
- `~/my.cnf`, for user-specific options.

Under Windows, without a `--defaults-file` argument, the MySQL server now looks for option files in *c:\windows*, then in *c:*, then in c*:\Program Files\MySQL\MySQL Server <version_number>*, looking in each place first for *my.ini*, then for *my.cnf*. The server looks for `defaults-extra-file` only if it is specified on the command line.

You have four ways of telling a MySQL program whether and how to read option files:

- `--no-defaults` causes no option files to be read (not recommended);
- `--defaults-file=fullPath` tells the MySQL program to read this option file*;*

- `--defaults-extra-file=fullPath` tells the server to read this file after the global option file but before the user configuration file;
- since version 4.11, `!include` *`fullpath`* in a section of an option file tells the group's program to include the file specified by *`fullpath`*, and `!includedir path` tells the group's program to search `path` for option files;

The argument `--print-defaults` prints the progam name and all options found and set. Command-line settings override option file settings. For any setting, the last one read is the one that sticks. Starting with 4.0.12 the server ignores world-writable configuration files. Under Windows use forward rather than backslashes in path arguments.

Table 3-2 shows the kinds of lines an option file can have. Use the `[client]` group header to group settings for MySQL client programs, and the `[mysqld]` group header for server programs. If your installation deposited sample configuration files for various server demand characteristics in `mysql/support-files`, one of them will likely serve as a good starter configuration file. When editing remember that MySQL programs ignore blank lines and leading and trailing blanks, and automatically translate the escape characters `\b`=backspace, `\t`=tab, `\n`=newline, `\r`=return, `\s`=space, `\\`=\.

Table 3-2: MySQL Option File Line Types

Line Type	***Description***
`comment`	*line beginning with # or /*
`[groupname]`	*header line which names a* `group`*, or the name of a program, whose options follow until another group header or EOF, whichever first occurs. Possible groups include* `[client]` *and* `[mysqld]` *(meaning server)*
`option`	*set* option *by name alone, (for example* `memlock` *to lock* `mysqld` *in memory), equivalent to* `--option` *on the command line*
`option=value`	*set* `option` *to* `value`*, equivalent to* `--option=value` *on the command line*
`varName=value`	*Equivalent to* `--varName=value` *on the command line; older* `set variable=` *prefix is deprecated.*

You have six ways to *read* MySQL system variables and their values (*Appendix B*):

- Some can be SELECTed in a MySQL client via `SELECT @@variableName`,
- Some are displayed in a MySQL client via `SHOW VARIABLES [LIKE wildspec]`,
- Some are displayed from the OS command line by `mysqld --help`,
- Some are displayed from the OS command line by `mysqladmin variables`,
- *MySQLAdministrator* lists many under `Startup Variables`.
- Query `information_schema`.

Most variables are displayed by one or more of these methods. A few are displayed by all methods, and a few by none. Most variables which can be `SELECT`ed can be set from a MySQL client prompt via `SET [GLOBAL | LOCAL] @@variableName syntax`. Variables that cannot be set in this way must be set in an option file, or on the command line.

An option that can be set in an option file can also be set on the command line by prepending a double dash `--` to the option. In some cases, MySQL also accepts a single-letter abbreviation for the command. For on/off options, if *`optionname`* can be set in an option file, it can also be set on the command line of the program that uses it with `--`*`optionname`*, `--enable-`*`optionname`* or `--`*`optionname`*`=1`, and can be disabled by `--skip-`*`optionname`*, `--disable-`*`optionname`* or `--`*`optionname`*`=0`. Except when you are explicitly tinkering with configuration settings, option files are preferable.

Which methods apply to which variables? If the variable name has hyphens, you cannot SELECT it; if it is SETtable, you usually can. But MySQL recognises some variable names both with hyphens and with underscores. And many variables are SETtable *and* SELECT-able. There are variables you would expect to be SELECTable, but are not. There are variables you can set in option files but are not listed by SHOW VARIABLES. And much of this is in flux, from version to version. All in all, there are no failsafe rules that reliably tell what method is sure to retrieve every configuration variable's value. Thus *Appendix B*, best printed out for quick reference.

MySQL client program

Command	*Abbrev*	*Meaniing*
?, help	\?, \h	Help
charset	\C	Set `charset`
clear	\c	Clear
delimiter	\d	Set statement delimiter
edit*	\e	Edit with `$editor`
ego	\G	Send command, display results vertically
exit, quit	\q	Exit
go	\g	Send command
nopager*	\n	Disable pager, print to `stdout`
pager*	\P	Print result to `pager`
print	\p	Print current command
prompt	\R	Set program prompt
rehash	\#	Rebuild completion hash
source†	\.	Execute SQL script file
status	\s	Server status
system*	\!	System shell command
tee	\T	Append all to `outfile`
use	\u	Use `database`
warnings	\W	Show warnings
nowarning	\w	Do not show warnings
** not in Windows*		*† no semicolon*

To execute MySQL commands, MySQL ships with the commandline client program *mysql*:

```
mysql [options] [databasename]
```

Specify `options` (*Table 3-3*) in *my.cnf/ini* under `[mysql]` without leading dashes, or on the commandline with leading double dashes or single dashes and option abbreviations. At least specify *server host* (server name or IP address), *username* and *password*, and server listening *port* if not the default 3306, *e.g.*:

```
mysql -hHOST -uUSR -pPWD
```

Automate that in a batch script. In *Nix, *mysql* saves commands to *.mysql_history* in your home folder, or to a file named by the environment variable `mysql_histfile`; setting that to `/dev/null` disables history-saving. In Windows, save history with `--tee=filepath`.

In Windows, the *mysql* client program runs better in a console replacement like *ConEmu*, which bypasses Windows console and Powershell mishandling of many standard charsets, *e.g.,* Japanese.

SQL scripts run from within the mysql client with the command, *without a semicolon* …

```
source scriptfilename
```

and from the operating system command line with either of:

```
mysql -escriptfilename
mysql <scriptfilename
```

both of which disable interactive mode. For more info on connecting to the server see *Chapter 6* (Connections and Sessions).

Table 3-3: MySQL Client Program Command Line Options

Command	*Abbrev*	*Meaning*
--auto-rehash [--disable-auto-rehash]		Enable automatic rehashing [Disable]
--auto-vertical-output		Display results vertically when too wide (since 5.5)

Command	***Abbrev***	***Meaning***
--batch	-B	Disable interaction & history file, enable `--silent`
--binary-as-hex		Show binary data as hex. Since 5.5.57, 5.6.37, 5.7.19
--binary-mode		Do not translate \n to \r\n; \0 not a terminator (since 5.6.3)
--compress	-C	Use compression in server-client protocol
--character-sets-dir=dirname		Directory holding character sets
--column-names		Show column names in result headers, default=on
--column-type-info	-m	Show column metadata. Since 5.1.4 (--debug-info, -T before)
--connect-expired-password		*Connect permitting password change commands only*
--connect-timeout=#		Set connection timeout
--database=name	-D	Use named database
--debug[=#]	-#	Debug or if non-debug version then exit
--debug-info	-T	Display debug info on exit
--default-character-set=name		Set default character set for MySQL client. Note: NOT 'character_set_client' as with server setting!
--delimiter=str		Use `str` as a command delimiter (`str` is case-sensitive)
--execute=scriptname --execute="query"	-e	Execute script and exit. Disables `--force`, history. Quoted script data must *not* have embedded ' chars.
--force	-f	Continue past SQL errors
--help	-?, -I	Display help and exit
--html	-H	Write output as HTML
--host=hostname	-h	Name of host to connect to, default `localhost`
--i-am-a-dummy	-U	Same as `--safe-updates`
--ignore spaces	-i	Ignore whitespace after function names
--init-command=stmt		Startup command, has not worked correctly for years
--line numbers		Number output lines
--local-infile=1|0		Enable or disable LOAD DATA LOCAL INFILE
--max_join_size=#		Max number of joined rows with `--safe-updates`
--max-packet-length=#		Maximum size of packet between client & server
--named-commands	-G	Enable client commands after `<cr>`, default=disabled.
--net-buffer-length=#		Size of TCP/IP or socket bujffer
--no-beep	-b	Disable error beeps
--one-database	-o	Update only the default database
--password	-p	Connection password, prompted for if left blank
--pipe	-W	Connect to server via named pipes
--prompt=txt		Set prompt, eg `\h.\d>` for `host.dbname`
--port=#	-P	Connect to server via port #.
--protocol=protocolname		Use this protocol for connection
--quick	-q	No result caching, disables history, slow if output suspended
--raw	-r	Use no escape conversion in output
--reconnect		Reconnect if connection is lost. Default=on
--safe-updates	-U	Allow UPDATEs and DELETEs that use keys
--secure-auth		Refuse connection using pre-4.1.1 authenticaton
--select_limit=#		Max SELECT size with `--safe-updates`, default=1000
--shared_memory_base_name=name		Set base name of shared memory
--show-warnings		Show warnings after each command (added in 5.0.6)
--silent	-s	Slim down output, tab-separate columns
--skip-column-names		Do not write column names in result headers
--skip-line-numbers		No line numbers on error
--socket=socketname	-S	Use named socket for connection
--ssl [--disable-ssl]		Enable SSL connection. [Disable]
--ssl-ca=name		CA file in PEM format
--ssl-ca-path=name		CA directory

Command	*Abbrev*	*Meaning*
--ssl-cert=name		X509 cert in PEM format
--ssl-cipher=name		SSL cipher
--ssl-key=name		X509 key in PEM format
--ssl-verify-server-cert		Verify server Common Name
--table	-t	Write output in ASCII tables, default=on
--tee=filename		Write everything to `filename` if not batch mode
--unbuffered	-n	Flush buffer after queries
--user=username	-u	Connect as `username`
--*varname*=value		Set variable named `varname` to `value`
--verbose	-v	Verbose output, -v -v -v for table format
--version	-V	Print version info and exit
--vertical	-E	Display query output rows vertically
--wait	-w	Wait and retry connection if it fails
--xml	-X	Write output as XML

Character sets and collation basics

A *character set* (default `Latin1`) defines characters for string values. *Collations* (default `latin1_swedish_ci`) define how values are compared and sorted, and associate with character sets (thus `Latin1`, `latin1_swedish_ci`); `_bin` at the end of the collation name means binary (bit-by-bit), and `_ci` at the end means case-insensitive. System variables determine character sets and collations (*Appendix B*, `char*`, `coll*`) for the server, USEd database and connections. Table, column and stored routine character sets and collations follow these defaults unless CREATE | ALTER statements specify otherwise. Retrieve available character sets and collations with …

```
SHOW CHARACTER SET;
SHOW COLLATION;
SELECT character_set_name, default_collate_name
FROM information_schema.character_sets;
```

To change available character sets and collations, recompile the server. For most purposes the best default is UTF8, so in *my.cnf/ini* you would write

```
[mysql]
default-character-set=utf8
[mysqld]
character-set-server=utf8
collation-server=utf8_unicode_ci
```

and you would begin client sesssions with

```
SET NAMES utf8;
```

To avoid *MySQL charset/collation hell*, get this right *before putting data in your tables*. MySQL recognises five levels of collation coercibility (*Chapter 8*, STRING FUNCTIONS, COERCIBILITY). Two rules govern collation differences (e.g., `colname=value`):

- the collation with lowest coercibility wins, and
- two sides with the same coercibility must have the same collation.

Character sets and collations

A *character set* (default `Latin1` before version 8, then `utf8mb4`) defines string character values. *Collations* (default `latin1_swedish_ci` before version 8, then `utf8mb4_0900_ai_ci`), define how character values compare, sort, and associate with charsets (*e.g.*, `Latin1`, `latin1_swedish_ci`); a collation name ending with `_bin` is binary (bit-by-bit), with `_ci` is case-insensitive. System variables set charsets and collations (*Appendix B*, `char*`, `coll*`) for the server, database in use, and connections. Table, column and stored routine charsets and collations follow these defaults unless CREATE | ALTER says otherwise. Retrieve available charsets and collations with …

```
SHOW CHARACTER SET;
SHOW COLLATION;
SELECT character_set_name, default_collate_name
FROM information_schema.character_sets;
```

To change available charsets and collations, recompile the server. To see your charset setitngs, issue `show variables like '%char%';` usually the best default is UTF8 or the version 8.0 default. In *my.cnf/ini* write

```
[mysql]
default_character_set=utf8mb4
[mysqld]
character_set_server=utf8
collation_server= utf8mb4_0900_ai_ci
```

and to be sure, you would begin client sesssions with

```
SET NAMES utf8mb4;
```

To avoid *MySQL charset/collation hell*, get this right *before adding data to tables*.

MySQL recognises five levels of collation coercibility (*Chapter 8*, STRING FUNCTIONS, COERCIBILITY). Two rules govern collation differences (*e.g.*, in `colname=value`):

- the collation with lowest coercibility wins, and
- two sides with the same coercibility must have the same collation.

TOC ***Previous*** ***Next***