Installing MySQL

A one-step for obtaining MySQL, *PHP*, *Perl*, Apache web server and *PHPMyAdmin* is to download and install *WampServer* or *XAMPP*. Does everything always work? *No*, and upgrades are a big problem with WAMP, but when they do work, they're very convenient. 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 fee.

There is a *Windows installer* to install and upgrade the MySQL server, *Connectors*, *WorkBench* and utilities. It's meant to be self-documenting and self-correcting but most published instances have had issues. If you come across such problems, end the installer task in *Task Manager*, open an administrator-level command prompt, navigate to your downloads folder, then run *msiexec /i* followed by the name of your *.msi* file. If it still hangs or crashes, report the relevant part of the installer log to the MySQL installer *forum* or *bug page*.

Use the step-by-steps in this chapter 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 ("drizzle" for a time, now defunct) while MySQL 5.5, 5.6 and 5.7 appeared. In April 2018 a Generally Available (GA) rel;ease of 8.0 appeared. So there have been six major releases since 5.0—two are active, four have been *archived*:

- ***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 archived..

- ***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. Reached end-of-life (EOL) 2013, archived.
- ***5.5***: GA release December 2010: added *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*. Reached EOL December 2018, archived.
- ***5.6***: GA release February 2013: crash-safe binary logs; PARTITION references in SELECT, INSERT, UPDATE and DELETE commands; UUIDs for each running server instance; delayed replication; replication row image control; PERFORMANCE_ SCHEMA enhancements; better optimisation of JOIN , WHERE and LIMIT; and expanded INFORMATION_SCHEMA INNODB metadata. Reached EOL February 2021, archived.
- ***5.7***: introduced April 2013, GA release October 2015: better security, logging and Triggers; subqueries in Views; JSON and derived columns; many bug fixes.
- ***8.0***: GA release Apr 2018, improves REGEX, moves system tables and the global data dictionary to InnoDB, revises partitioning; bit functions now do binary strings; enhances query parsing and indexing; adds component-based infrastructure, SQL roles, `utf8mbr` collation; recursive Common Table Expressions (CTEs); persistent dynamic global variables. For dozens of server team articles look for "8.0" *here*.

A *serious password-hacking vulnerability* renders all MySQL 5.x versions 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*: version 5.5 is drop-in compatible with MySQL 5.5; and versions 10.0 (2017) through 10.5 (2020) mostly match MySQL 5.6 through 8.0 while adding innovations including alternative storage engines. The online manual is *here*. Non-commercial user support is mainly via mailing list rather than MySQL's multiple *MySQL fora*.

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 why 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 5.5 and 5.6 packages included *NDB CLUSTER*; now it's *separate*. In practice, MySQL-OS matchings (Table 3-1) can vary from what MySQL reports. 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 Linux varieties 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, there are full versions with and without Windows installer; avoid the skeletal Essentials package. Since 5.5.11, the embedded server is not in the *.msi* package.

Like package content, installation details change from release to release. 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 management, security, performance, debugging, response to SQL syntax, etc. These can be set on the 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 apt repo			•	•				•	•	•	•
Debian Linux 4			•	•		•	•				
Debian Linux 5			•	•			•	•			
Debian Linux 6			•	•			•	•	•		
Debian Linux 7, 8			•	•					•	•	
Debian GNU/Linux 9			•	•					•	•	•
Fedora Yum Repo			•	•				•	•	•	•
FreeBSD 11.1+				•					•	•	•
Generic Linux .tar			•	•				•	•	•	•
Oracle Linux 4			•	•	•	•	•	•			
Oracle Linux 5			•	•		•	•	•	•	•	
Oracle Linux 6			•	•				•	•	•	
Oracle Linux 7				•				•	•	•	•
OS X 10.5			•	•			•	•			
OS X 10.6			•	•			•	•	•		
OS X 10.7, 10.8				•					•		
OS X 10.9, 10.10				•					•	•	
OS X 10.11-14				•						•	•
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 11+)		•	•	•		•	•	•	•		
Solaris 11 update 4+		•		•				•	•	•	•
SUSE Ent Linux 9				•	•	•	•				
SUSE Ent Linux 10				•	•	•	•	•			
SUSE Ent Linux 11				•			•	•	•		
SUSE Ent Linux 12				•					•	•	•
SUSE Ent Linux 15				•							•
SUSE repo			•	•				•	•	•	•
Ubuntu 12.04/14.04 LTS			•	•				•	•	•	
Ubuntu 16.04/18.04 LTS			•	•						•	•
Windows 2008 Server R2				•			•	•	•	•	
Windows 2012 Server R2				•				•	•	•	•
Windows 2016 Server				•					•	•	•
Windows 2019 Server				•							•
Windows 7			•	•			•	•	•	•	•
Windows 8			•	•				•	•	•	•
Windows 10			•	•						•	•

Authentication

Traditionally, MySQL authenticated users by just matching a user's connect address (`host`), name (`user`) and hashed `password` to a `mysql.user` row created directly or with GRANT, unless the server starts with `--skip-grant-tables`. But …

1. Since MySQL 5.5 the non-nullable `mysql.user.plugin` column must specify an authentication *plugin*. Before 8.0 `mysql_native_password` was the default value; since, `caching_sha2_password` is the default because it supports SSL (*caching_sha256_password plugin* in MariaDB). MySQL passes the value found in `mysql.authentication_string` to the plugin, which may also return a *proxy username* defined by CREATE USER or GRANT. Validity is per-connection. By default MySQL 5.7 and 8.0 *enable SSL*; require SSL encryption for all logins with `require_secure_ transport=ON` in *my.cnf/ini*, for a user with CREATE USER…REQUIRE ssl (*Chapter 6*). More documentation is in *Chapter 19* (Configuring databases for security).

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

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. Once `login_path_name` specs are stored, invoke the *mysql* client program with …

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

5. Version 8 has implemented 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,which change often, 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 first for *my.ini*, then for *my.cnf*, in *c:\windows*, *c:*, then in c:*\Program Files\MySQL\ MySQL Server <version_number>*; it looks for `defaults-extra-file` only if specified.

You have four ways to tell 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 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` lists progam name and options. Command-line settings override option file settings. For any setting, the last one read sticks. The server ignores—with a mysql error log warning—a world-writable option file, and *silently* ignores an option file if the server lacks read permission for it! Use forward slashes in Windows paths.

Table 3-2 lists options file line types. Use the `[client]` group header to group settings for MySQL client programs, and the `[mysqld]` group header for server programs. If your installation put sample configuration files for various server demand characteristics in `mysql/support-files`, one of them will likely serve as a good starter configuration file. 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 `SELECT`able variables 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 its name. Some have single-letter abbreviations. On/off options can be set with *`optionname`*, `enable-`*`optionname`* or *`optionname`*`=1` and unset with `skip-`*`optionname`*, `disable-`*`optionname`* or *`optionname`*`=0`.

Which methods apply to which variables? Obviously a vairiable with a hyphen in its name will be awkward to SELECT, so selectable hyphenated variables have alternate names in which underscores replace the hyphens. There's no failsafe rule that can indicate reliably what method is sure to retrieve a given variable's value. Keep a *link* to thevariables page for your MySQL version.

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	Do cmd, vertical results
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
ssl*		(Table 3-3.)
status	\s	Server status
system**	\!	System shell command
tee	\T	Append all to `outfile`
use	\u	Use `database`
warnings	\W	Auto-show warnings
nowarning	\w	Do not show warnings

** not Windows ** Windows since 8.0.19 † 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 command-line 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 the 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*, bypassing Windows console and Powershell mishandling of many standard charsets, *e.g.*, Japanese.

SQL scripts run in the mysql client with:

```
source full_path_to scriptfilename
```

with no terminating semicolon, and from the commandline with:

```
mysql –efull_path_to_scriptfilename
mysql <full_path_to_scriptfilename
```

both of which disable interactive mode. See also *Chapter 6* (Connections and Sessions).

Table 3-3: MySQL Client Program Command Line Options

Command	*Abbrev*	*Meaning*
--[disable-]auto-rehash		Enable automatic rehashing [Disable]
--auto-vertical-output		Display results vertically when too wide (since 5.5)
--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.
--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		Default character set (NOT ‘character_set_client’)
--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

Command	*Abbrev*	*Meaning*
--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="statement(s)"		Startup command(s), terminate each with semicolon
--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	Do not cache results. Less memory needed, no history
--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 `--safe-updates` statement size, default 1000
--shared_memory_base_name=name		Set base name of shared memory
--show-warnings		Show warnings after each command
--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]		En/disable SSL connection. 8.0: use ssl-mode-REQUIRED
--ssl-ca=name		CA file in PEM format
--ssl-ca-path=name		CA directory
--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. Removed 8.0, use ssl-mode=VERIFY_IDENTITY
--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

MySQL Shell and document engine

IIn April 2016 MySQL introduced *MySQL Shell* (*mysqlsh*), a command-line client and code editor that can run in SQL, Javascript or Python mode. Along with it came an *X Server Plugin*

for storage of, and CRUD operations on, a document store of JSON objects, and an *X DevAPI* for relational and document data. *Mysqlsh* communicates with a MySQL Server via both SQL and the new *X Protocol*. Documentation for the shell is *here*, for the document store *here*.

As an alternative commandline client, *mysqlsh* is not just much more versatile, in SQL mode it runs queries about 50% faster than the legacy MySQL client program, and can *speed up database dump and load* operations considerably.

Character set and collation basics

A *character set* defines how characters map to 0s and 1s. Before 8.0 the MySQL default was `Latin1`; since 8.0.1 the default is `utf8mb4`, which is *fully-4-byte `UTF-8`*. *Collations* (default `latin1_swedish_ci` before version 8, since 8.0.1 `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), `_ci` indicates case-insensitivity. System variables set charsets and collations (*Appendix B*, `char*`, `coll*`) for the server, databases, and connections. Table, column and stored routine charsets and collations follow these defaults except when 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, the version 8.0 default. In *my.cnf/ini* write …

```
[mysql]
default_character_set=utf8mb4
[mysqld]
character_set_server=utf8mb4
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*.

There are 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.

Installing a binary distribution under Linux

If you are upgrading, work out your *upgrade strategy* before you begin. Do a complete data backup using *mysqldump*, e.g. …

```
mysqldump –uUSR –pPWD –K –E –A –R –f >some_backup_dir/mybackup.sql
```

If upgrading to 5.7.2 or later, and if the dump file will be loaded into the new installation, add `–skip-flush_privileges` to the above command. Then shut down the server and save copies of the server program and *my.cnf*

From the Linux prompt

It's easier than it used to be…: