

# 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 engine](#)  
[Connector/ODBC](#) [MySQL Workbench](#)

The easiest way to set yourself up with MySQL, *PHP*, *Perl* and PhpMyAdmin all at once is to download and install *XAMPP*. No hassles, and everything works. The rest of this chapter is for those who want to install MySQL and APIs individually.

MySQL provides *binary* and *source* distributions. Unless you need to customise MySQL code and you have considerable experience with C/C++, go with a binary distribution, installation of which comes down to five main steps: choose a package for your operating system, download the package, unpack/install it, configure it, and test it.

## What version, what operating system?

If you develop MySQL databases for deployment elsewhere, you need an installation of each current production release on your development computer(s). All production releases branch to free *Community* and pay-by-subscription *Enterprise Server* editions.

Extended support for version 4.1 ended 31 Dec 2009. Version 5.0 went into production in 2005, 5.1 in 2008, 5.5 in late 2010. Version 6.0 appeared in 2007, but in 2009 MySQL announced a *new “milestone” release model*, withdrew 6.0, and said a new 6.0 would reappear in 2010 following a new 5.7 with features from the withdrawn 6.0. Instead, 5.5 and 5.6 have appeared. Currently, then, there are four supported release tracks:

- **5.0** added Stored Routines, Triggers, Views, `information_schema`, XA transactions, NDB CLUSTER and FEDERATED engines. Active 5.0 development ended 31 Dec 2009; it is available now as an *archived release*.
- **5.1**: adds *partitions*, *an event scheduler*, *row-based replication and more logs*, a *plugin API*, many NDB CLUSTER storage engine improvements, a *load emulator* and more `information_schema` tables. It accepts the blazingly fast, ACID-compliant *PBXT* storage engine plugin. Active development ends 31 Dec 2010.
- **5.5** is now a general availability release; it includes the *new INNODB engine* with concurrency, thread and lock improvements; *SIGNAL*, *RESIGNAL*; *LOAD XML and partitioning extensions*; *semisynchronous replication*; performance enhancements

for Solaris; *DTrace* monitoring on some Solaris and MAC systems, and the interesting *new authentication plugin architecture*.

- **5.6** was introduced in April 2011. It adds crash-safe binary logs; PARTITION references in SELECT, INSERT, UPDATE and DELETE commands; UUIDs for every running MySQL 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.

How well MySQL performs on a particular operating system (OS) depends on:

- quality of the OS kernel and the file system,
- stability of the thread library,
- capacity of the kernel to use shared multi-processing,
- thread lock/unlock flexibility, and
- stability and robustness of a given OS build.

Thanks to 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 is MySQL for Linux more popular than MySQL for Linux? No. As of mid-2007, MySQL for Windows was downloaded *three times more often* than MySQL for Linux. Why? The Windows user interface is richer. The moral of the story seems to be: develop MySQL databases on Windows, and deploy them on \*Nix.

The *bug list* for 5.1 at the time of its General Availability release had more than 300 non-trivial entries. Version 5.5 production releases have been more stable.

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 package definitions *change often*. There have been confusing version-to-version changes in relationships amongst package names, server names and *storage engines*, for example

- **Linux**: the *mysqld* server used to support only MYISAM, INNODB, MEMORY and MERGE tables. Before 4.0, *mysqld-max* was needed for INNODB. From 4.0 to 5.1.5 you needed *mysqld-max* for BDB and NDB CLUSTER, and in 5.1.3 for partitioning.
- **Windows**: until 5.0.25 and 5.1.12 there was a *mysqld.exe* build for MYISAM, INNODB, MEMORY and MERGE tables. From 4.0 through 5.0.24/5.1.5 there was a *mysqld-max.exe* build for BDB and partitioning. In addition, until 5.1.21 only *mysqld-nt.exe* supported named pipes. Since 5.1.21, all these functionalities are included in *mysqld.exe*. A *Windows binary with NDB* has also appeared.

In general, more recent MySQL releases are available only for recent operating system versions. Make your choice *here*, then download from *here* (earlier builds *here*). There are four *main package patterns*:

- for Linux: *RedHat Package Manager* (RPM) bundles containing the server and client, plus bundles containing NDB, and 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*, though, and it plays back into installation, so you need to know at least a little about configuration before you start. There are *more than 500* configuration options that govern MySQL character sets and collations, memory use, file location, transaction systems and behaviour, security, performance, debugging, response to SQL syntax, and other server behaviours. 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`. You can come back later for details.

## Authentication

MySQL servers have always authenticated by matching the user's connection address (host), username (`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 the `--skip-grant-tables` option. Since 5.5.7, authentication may also use *custom plugins* listed in two new columns of `mysql.user`, `plugin` (plugin name) 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).

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'`.

Since 5.5.16, MySQL Enterprise ships with server-side Pluggable Authentication Modules (PAM) for Mac OS and Linux and for Windows 2000 and 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.

## 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. In a few cases these sources disagree with one another or with server behaviour.

System variables, their names, and their syntax are in flux. Thus *Appendix B*, which lists the 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;
- in version 4 since 4.11, and in 5 since 5.0.4, *!include fullPath* in a section of an option file tells the group's program to include the file specified by *path*, and *!includedir path* tells the group's program to search *path* for option files.

The argument *--print-defaults* prints the program 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 3-1: MySQL Option File Line Types**

<b>Line Type</b>	<b>Description</b>
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.

Table 3-1 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, \\=\.

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 SELECTed 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.

There are rules that may help determine which method to use to read or set a variable's value. For example, 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. Also, there are variables which are both SETtable and SELECTable, yet the documentation does not say so. There are variables you would expect to be SELECTable, but are not. There are variables you can set in option files but which

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.

## Character sets and collations

The *character set* (default `Latin1`) defines characters for storing values. *Collation* (default `latin1_swedish_ci`) defines how values are compared and sorted. Collations 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. There are system variables for setting available and default character sets and collations for the server, the default database, and connections (*Appendix B*, `character set*`, `collation*`). You can specify the character set and collation for a particular database, table or column when you create it or alter it (*Chapter 6*, `CREATE DATABASE`, `ALTER DATABASE`, `CREATE TABLE`, `ALTER TABLE`). `SHOW CHARACTER SET` and `SHOW COLLATION` (*Chapter 6*, `SHOW`) list available character sets and collations. Version 5.0.3 introduced a `SELECT` syntax for such queries (*Chapter 6*, `INFORMATION_SCHEMA`). To change available and default character sets and collations, recompile the server.

Since 5.0.3 MySQL recognises five levels of collation coercibility (*Chapter 8*, `STRING FUNCTIONS`, `COERCIBILITY`) and uses these rules to resolve collation conflicts:

- in an expression, for example `colname = value`, use the collation with the lowest coercibility value;
- it is an error if the two sides of an expression have the same coercibility but different collations.

## MySQL client program

MySQL ships with the client program *mysql* for executing MySQL commands:

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

You can specify *mysql* options (Table 3-2) in the `[mysql]` section of *my.cnf/ini*, without leading dashes, or on the command line with leading double dashes or single dashes and option abbreviations. At a minimum, you generally need to specify the *server host* (the name or IP address of the server), your *username*, your *password*, and the *port* on which the server is listening if that differs from the default of 3306, for example:

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

You will be executing that command often, so automate it in a script. Within *mysql*, `?` and `help` list *available non-SQL commands* and abbreviations, though not all work as documented under Windows. Under \*nix, *mysql* saves commands to *.mysql\_history* in the user's home directory, or to a file named by the environment variable `mysql_histfile`, so setting that variable to `/dev/null` disables history saving.

**Table 3-2: MySQL Client Program Command Line Options**

Command	Abbrev	Meaning
<code>--auto-rehash</code> [ <code>--disable-auto-rehash</code> ]		Enable automatic rehashing [Disable]

<b>Command</b>	<b>Abbrev</b>	<b>Meaning</b>
--auto-vertical-output		Display results vertically when too wide (since 5.5)
--batch	-B	Disable interaction & history file, enable --silent
--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-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 <i>not</i> 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 buffer
--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. Disables history, may slow down the server if output is 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 authentication
--select-limit=#		Max SELECT statement 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

<i>Command</i>	<i>Abbrev</i>	<i>Meaning</i>
--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

*To read the rest of this and other chapters, buy a copy of the book*

---

*[TOC](#) [Previous](#) [Next](#)*

---