Building MySQL from Source

Abstract

This is the Building MySQL from Source extract from the MySQL 5.1 Reference Manual.

For help with using MySQL, please visit either the MySQL Forums or MySQL Mailing Lists, where you can discuss your issues with other MySQL users.

For additional documentation on MySQL products, including translations of the documentation into other languages, and downloadable versions in variety of formats, including HTML and PDF formats, see the MySQL Documentation Library.

Document generated on: 2016-08-12 (revision: 48536)

Table of Contents

Preface and Legal Notices

This is the Building MySQL from Source extract from the MySQL 5.1 Reference Manual.

Legal Notices

Documentation Accessibility

For information about Oracle's commitment to accessibility, visit the Oracle Accessibility Program website at

http://www.oracle.com/pls/topic/lookup?ctx=acc&id=docacc.

Access to Oracle Support

Oracle customers that have purchased support have access to electronic support through My Oracle Support. For information, visit http://www.oracle.com/pls/topic/lookup?ctx=acc&id=info or visit http://www.oracle.com/pls/topic/lookup?ctx=acc&id=trs if you are hearing impaired.

Chapter 1 Installing MySQL from Source

Building MySQL from the source code enables you to customize build parameters, compiler optimizations, and installation location. For a list of systems on which MySQL is known to run, see http://www.mysql.com/support/supportedplatforms/database.html.

Before you proceed with an installation from source, check whether we produce a precompiled binary distribution for your platform and whether it works for you. We put a great deal of effort into ensuring that our binaries are built with the best possible options for optimal performance. Instructions for installing binary distributions are available in Installing MySQL on Unix/Linux Using Generic Binaries.

To obtain a source distribution for MySQL, see How to Get MySQL. MySQL source distributions are available as compressed `tar` files, Zip archives, or RPM packages. Distribution files have names of the form `mysql-VERSION.tar.gz`, `mysql-VERSION.zip`, or `mysql-VERSION.rpm`, where *`VERSION`* is a number like `5.1.73`.

To perform a MySQL installation using the source code:

- To build MySQL from source on Unix-like systems, including Linux, commercial Unix, BSD, OS X and others using a `.tar.gz` or RPM-based source code distribution, see Chapter 2, *Installing MySQL Using a Standard Source Distribution*.
- To build MySQL from source on Windows (Windows XP or newer required), see Chapter 6, *Installing MySQL from Source on Windows*.
- For information on building from one of our development trees, see Chapter 3, *Installing MySQL Using a Development Source Tree*.
- For information on using the `configure` command to specify the source build parameters, including links to platform specific parameters that you might need, see Chapter 4, *MySQL Source-Configuration Options*.

To install MySQL from source, the following system requirements must be satisfied:

- GNU `gunzip` to uncompress the distribution and a reasonable `tar` to unpack it (if you use a `.tar.gz` distribution), or `WinZip` or another tool that can read `.zip` files (if you use a `.zip` distribution).

  GNU `tar` is known to work. The standard `tar` provided with some operating systems is not able to unpack the long file names in the MySQL distribution. You should download and install GNU `tar`, or if available, use a preinstalled version of GNU tar. Usually this is available as `gnutar`, `gtar`, or as `tar` within a GNU or Free Software directory, such as `/usr/sfw/bin` or `/usr/local/bin`. GNU `tar` is available from http://www.gnu.org/software/tar/.
- A working ANSI C++ compiler. GCC 3.4.6 or later, Sun Studio 10 or later, Visual Studio 2005 or later, and many current vendor-supplied compilers are known to work.
- A good `make` program. Although some platforms come with their own `make` implementations, it is highly recommended that you use GNU `make` 3.75 or newer. It may already be available on your system as `gmake`. GNU `make` is available from http://www.gnu.org/software/make/.
- `libtool` 1.5, available from http://www.gnu.org/software/libtool/. 1.5.24 or later is recommended.

If you run into problems and need to file a bug report, please use the instructions in How to Report Bugs or Problems.

Chapter 2 Installing MySQL Using a Standard Source Distribution

To install MySQL from source, first configure, build, and install from a source package. Then follow the same postinstallation setup sequence as for a binary installation.

If you start from a source RPM, use the following command to make a binary RPM that you can install. If you do not have `rpmbuild`, use `rpm` instead.

```
shell> rpmbuild --rebuild --clean MySQL-VERSION.src.rpm
```

The result is one or more binary RPM packages that you install as indicated in Installing MySQL on Linux Using RPM Packages.

The sequence for installation from a compressed `tar` file source distribution is similar to the process for installing from a generic binary distribution that is detailed in Installing MySQL on Unix/Linux Using Generic Binaries. For a MySQL `.tar.gz` source distribution, the basic installation command sequence looks like this:

```
# Preconfiguration setup
shell> groupadd mysql
shell> useradd -g mysql -s /bin/false mysql
# Beginning of source-build specific instructions
shell> tar zxvf mysql-VERSION.tar.gz
shell> cd mysql-VERSION
shell> ./configure --prefix=/usr/local/mysql
shell> make
shell> make install
# End of source-build specific instructions
# Postinstallation setup
shell> cd /usr/local/mysql
shell> chown -R mysql .
shell> chgrp -R mysql .
shell> bin/mysql_install_db --user=mysql
shell> chown -R root .
shell> chown -R mysql var
# Next command is optional
shell> cp support-files/my-medium.cnf /etc/my.cnf
shell> bin/mysqld_safe --user=mysql &
# Next command is optional
shell> cp support-files/mysql.server /etc/init.d/mysql.server
```

A more detailed version of the source-build specific instructions is shown following. Perform the following steps as the `mysql` user, except as noted.

> **Note**
>
> The procedure shown here does not set up any passwords for MySQL accounts. After following the procedure, proceed to Postinstallation Setup and Testing, for postinstallation setup and testing.

1. Set up the `mysql` user and group that will be used to run and execute the MySQL server and own the database directory. For details, see Creating a `mysql` System User and Group, in Installing MySQL on Unix/Linux Using Generic Binaries.

2. Pick the directory under which you want to unpack the distribution and change location into it.

3. Obtain a distribution file using the instructions in How to Get MySQL.

4. Unpack the distribution into the current directory. `tar` can uncompress and unpack the distribution if it has `z` option support:

   ```
   shell> tar zxvf /path/to/mysql-VERSION.tar.gz
   ```

   This command creates a directory named `mysql-VERSION`.

   If your `tar` does not have `z` option support, use `gunzip` to unpack the distribution and `tar` to unpack it:

   ```
   shell> gunzip < /path/to/mysql-VERSION.tar.gz | tar xvf -
   ```

5. Change location into the top-level directory of the unpacked distribution:

   ```
   shell> cd mysql-VERSION
   ```

6. Configure the source directory:

   ```
   shell> ./configure --prefix=/usr/local/mysql
   ```

   When you run `configure`, you might want to specify other options. For example, if you need to debug `mysqld` or a MySQL client, run `configure` with the `--with-debug` option, and then recompile and link your clients with the new client library. See Debugging and Porting MySQL.

   Run `./configure --help` for a list of options. Chapter 4, *MySQL Source-Configuration Options*, discusses some of the more useful options.

   If `configure` fails and you are going to send mail to a MySQL mailing list to ask for assistance, please include any lines from `config.log` that you think can help solve the problem. Also include the last couple of lines of output from `configure`. To file a bug report, please use the instructions in How to Report Bugs or Problems.

7. Compile the source distribution:

   ```
   shell> make
   ```

   Use `gmake` instead on systems where you are using GNU `make` and it has been installed as `gmake`.

   If the compile fails, see Chapter 5, *Dealing with Problems Compiling MySQL*, for help.

8. Install the distribution:

   ```
   shell> make install
   ```

   You might need to run this command as `root`.

The remainder of the installation process, including setting up the configuration file, creating the core databases, and starting the MySQL server, are identical to the remainder of the process as shown in Generic Binary Install.

After everything has been installed, test the distribution. To start the MySQL server, use the following command:

```
```

```
shell> /usr/local/mysql/bin/mysqld_safe --user=mysql &
```

If you run the command as `root`, you should use the `--user` option as shown. The option value is the name of the login account that you created in the first step to use for running the server. If you run the `mysqld_safe` command while logged in as that user, you can omit the `--user` option.

If the command fails immediately and prints `mysqld ended`, look for information in the error log (which by default is the *`host_name`*`.err` file in the data directory).

For more information about `mysqld_safe`, see `mysqld_safe` — MySQL Server Startup Script.

To make it more convenient to invoke programs installed in `/usr/local/mysql/bin`, you can add that directory to your `PATH` environment variable setting. That enables you to run a program by typing only its name, not its entire path name. See Setting Environment Variables.

> **Note**
>
> The accounts that are listed in the MySQL grant tables initially have no passwords. After starting the server, you should set up passwords for them using the instructions in Postinstallation Setup and Testing.

Chapter 3 Installing MySQL Using a Development Source Tree

This section discusses how to install MySQL from the latest development source code.

To obtain the source tree, you must have Bazaar installed. The Bazaar VCS Web site has instructions for downloading and installing Bazaar on different platforms. Bazaar is supported on any platform that supports Python, and is therefore compatible with any Linux, Unix, Windows, or OS X host.

MySQL development projects are hosted on Launchpad. MySQL projects, including MySQL Server, MySQL Workbench, and others are available from the Oracle/MySQL Engineering page. For the repositories related only to MySQL Server, see the MySQL Server page.

To build under Unix/Linux, you must have the following tools installed:

- A good `make` program. Although some platforms come with their own `make` implementations, it is highly recommended that you use GNU `make` 3.75 or newer. It may already be available on your system as `gmake`. GNU `make` is available from http://www.gnu.org/software/make/.
- `autoconf` 2.58 (or newer), available from http://www.gnu.org/software/autoconf/.
- `automake` 1.8.1, available from http://www.gnu.org/software/automake/.
- `libtool` 1.5, available from http://www.gnu.org/software/libtool/. 1.5.24 or later is recommended.
- `m4`, available from http://www.gnu.org/software/m4/.
- `bison` 2.1 or newer, available from http://www.gnu.org/software/bison/. (Version 1 is no longer supported.) Use the latest version of `bison` where possible; if you experience problems, upgrade to a later version, rather than revert to an earlier one.

To build under Windows you must have Microsoft Visual C++ 2005 Express Edition, Visual Studio .Net 2003 (7.1), or Visual Studio 2005 (8.0) compiler system.

Once the necessary tools are installed, create a local branch of the MySQL development tree on your machine using this procedure:

1. To obtain a copy of the MySQL source code, you must create a new Bazaar branch. If you do not already have a Bazaar repository directory set up, you must initialize a new directory:

   ```
   shell> mkdir mysql-server
   shell> bzr init-repo --trees mysql-server
   ```

   This is a one-time operation.

2. Assuming that you have an initialized repository directory, you can branch from the public MySQL server repositories to create a local source tree. To create a branch of a specific version:

   ```
   shell> cd mysql-server
   shell> bzr branch lp:mysql-server/5.1 mysql-5.1
   ```

   This is a one-time operation per source tree. You can branch the source trees for several versions of MySQL under the `mysql-server` directory.

3. The initial download will take some time to complete, depending on the speed of your connection. Please be patient. Once you have downloaded the first tree, additional trees should take significantly less time to download.

4. When building from the Bazaar branch, you may want to create a copy of your active branch so that you can make configuration and other changes without affecting the original branch contents. You can achieve this by branching from the original branch:

```
shell> bzr branch mysql-5.1 mysql-5.1-build
```

5. To obtain changes made after you have set up the branch initially, update it using the `pull` option periodically. Use this command in the top-level directory of the local copy:

```
shell> bzr pull
```

   To examine the changeset comments for the tree, use the `log` option to `bzr`:

```
shell> bzr log
```

   You can also browse changesets, comments, and source code online at the Launchpad MySQL Server page.

   If you see diffs (changes) or code that you have a question about, do not hesitate to send email to the MySQL `internals` mailing list. See MySQL Mailing Lists. If you think you have a better idea on how to do something, send an email message to the list with a patch.

After you have the local branch, you can build MySQL server from the source code. On Windows, the build process is different from Unix/Linux: see Chapter 6, *Installing MySQL from Source on Windows*.

On Unix/Linux, use the `autoconf` system to create the `configure` script so that you can configure the build environment before building. The following example shows the typical commands required to build MySQL from a source tree.

1. Change location to the top-level directory of the source tree; replace `mysql-5.1` with the appropriate directory name.

```
shell> cd mysql-5.1
```

2. Prepare the source tree for configuration.

   Prior to MySQL 5.1.12, you must separately configure the `InnoDB` storage engine. Run the following command from the main source directory:

```
shell> (cd storage/innobase; autoreconf --force --install)
```

   You can omit the previous command for MySQL 5.1.12 and later, or if you do not require `InnoDB` support.

   Prepare the remainder of the source tree:

```
shell> autoreconf --force --install
```

   As an alternative to the preceding `autoreconf` command, you can use `BUILD/autorun.sh`, which acts as a shortcut for the following sequence of commands:

```
shell> aclocal; autoheader
shell> libtoolize --automake --force
```

```
shell> automake --force --add-missing; autoconf
```

If you get some strange errors during this stage, verify that you have the correct version of `libtool` installed.

3. Configure the source tree and compile MySQL:

```
shell> ./configure  # Add your favorite options here
shell> make
```

For a description of some `configure` options, see Chapter 4, *MySQL Source-Configuration Options*.

A collection of configuration scripts is located in the `BUILD/` subdirectory. For example, you may find it more convenient to use the `BUILD/compile-pentium-debug` script than the preceding set of shell commands. To compile on a different architecture, modify the script by removing flags that are Pentium-specific, or use another script that may be more appropriate. These scripts are provided on an "as-is" basis. They are not supported and their contents may change from release to release.

4. When the build is done, run `make install`. Be careful with this on a production machine; the installation command may overwrite your live release installation. If you already have MySQL installed and do not want to overwrite it, run `./configure` with values for the `--prefix`, `--with-tcp-port`, and `--with-unix-socket-path` options different from those used by your production server. For additional information about preventing multiple servers from interfering with each other, see Running Multiple MySQL Instances on One Machine.

5. Play hard with your new installation. For example, try to make new features crash. Start by running `make test`. See The MySQL Test Suite.

6. If you have gotten to the `make` stage, but the distribution does not compile, please enter the problem into our bugs database using the instructions given in How to Report Bugs or Problems. If you have installed the latest versions of the required tools, and they crash trying to process our configuration files, please report that also. However, if you get a `command not found` error or a similar problem for required tools, do not report it. Instead, make sure that all the required tools are installed and that your `PATH` variable is set correctly so that your shell can find them.

Chapter 4 MySQL Source-Configuration Options

The `configure` script provides a great deal of control over how you configure a MySQL source distribution. Typically, you do this using options on the `configure` command line. For a full list of options supported by `configure`, run this command:

```
shell> ./configure --help
```

You can also affect `configure` using certain environment variables. See Environment Variables.

The following table shows the available `configure` options.

Table 4.1 MySQL Source-Configuration Option Reference (`configure`)

Formats	Description	Default	Introduced	Removed
`--bindir`	User executables	`EPREFIX/bin`		
`--build`	Configure for building on BUILD	`guessed`		
`--cache-file`	Cache test results in FILE	`disabled`		
`--config-cache`	Alias for `--cache-file=config.cache'			
`--datadir`	Read-only architecture-independent data	`PREFIX/share`		
`--disable-FEATURE`	Do not include FEATURE			
`--disable-community-features`	Disable additional features provided by the community		5.1.28	
`--disable-dependency-tracking`	Disable dependency tracking			
`--disable-grant-options`	Disable GRANT options			
`--disable-largefile`	Omit support for large files			
`--disable-libtool-lock`	Disable libtool lock			
`--disable-thread-safe-client`	Compile the client without threads		5.1.7	
`--enable-FEATURE`	Enable FEATURE			
`--enable-assembler`	Use assembler versions of some string functions if available			
`--enable-debug-sync`	Compile in Debug Sync facility		5.1.41	
`--enable-dependency-tracking`	Do not reject slow dependency extractors			
`--enable-fast-install`	Optimize for fast installation	`yes`		
`--enable-local-infile`	Enable LOCAL for LOAD DATA INFILE	`disabled`		
`--enable-profiling`	Build a version with query profiling code		5.1.24	
`--enable-shared`	Build shared libraries	`yes`		

Formats	Description	Default	Introduced	Removed
`--enable-static`	Build static libraries	`yes`		
`--enable-thread-safe-client`	Compile the client with threads			5.1.6
`--exec-prefix`	Install architecture-dependent files in EPREFIX			
`--help`	Display help message and exit			
`--host`	Cross-compile to build programs to run on HOST			
`--includedir`	C header files	`PREFIX/include`		
`--infodir`	Info documentation	`PREFIX/info`		
`--libdir`	Object code libraries	`EPREFIX/lib`		
`--libexecdir`	Program executables	`EPREFIX/libexec`		
`--localstatedir`	Modifiable single-machine data	`PREFIX/var`		
`--mandir`	man documentation	`PREFIX/man`		
`--no-create`	Do not create output files			
`--oldincludedir`	C header files for non-gcc	`/usr/include`		
`--prefix`	Install architecture-independent files in PREFIX			
`--program-prefix`	Prepend PREFIX to installed program names			
`--program-suffix`	Append SUFFIX to installed program names			
`--program-transform-name`	run sed PROGRAM on installed program names			
`--quiet`	Do not print `checking...' messages			
`--sbindir`	System administrative executables	`EPREFIX/sbin`		
`--sharedstatedir`	Modifiable architecture-independent data	`PREFIX/com`		
`--srcdir`	Find the sources in DIR	`configure directory or ..`		
`--sysconfdir`	Read-only single-machine data	`PREFIX/etc`		
`--target`	Configure for building compilers for TARGET			
`--version`	Display version information and exit			
`--with-PACKAGE`	Use PACKAGE			
`--with-archive-storage-engine`	Enable the Archive Storage Engine	`no`		5.1.9

Formats	Description	Default	Introduced	Removed
--with-atomic-ops	Implement atomic operations using pthread rwlocks or atomic CPU instructions for multi-processor		5.1.12	
--with-berkeley-db	Use BerkeleyDB located in DIR	no		5.1.11
--with-berkeley-db-includes	Find Berkeley DB headers in DIR			5.1.11
--with-berkeley-db-libs	Find Berkeley DB libraries in DIR			5.1.11
--with-big-tables	Support tables with more than 4 G rows even on 32 bit platforms			
--with-blackhole-storage-engine	Enable the Blackhole Storage Engine	no		5.1.9
--with-charset	Default character set			
--with-client-ldflags	Extra linking arguments for clients			
--with-collation	Default collation			
--with-comment	Comment about compilation environment			
--with-csv-storage-engine	Enable the CSV Storage Engine	yes		5.1.9
--with-darwin-mwcc	Use Metrowerks CodeWarrior wrappers on OS X/Darwin			
--with-debug	Add debug code (optionally with memory checker, very slow)		5.1.7	
--with-embedded-privilege-control	Build parts to check user's privileges (only affects embedded library)			
--with-embedded-server	Build the embedded server			
--with-error-inject	Enable error injection in MySQL Server		5.1.11	
--with-example-storage-engine	Enable the Example Storage Engine	no		5.1.9
--with-extra-charsets	Use charsets in addition to default			
--with-fast-mutexes	Compile with fast mutexes	disabled	5.1.5	
--with-federated-storage-engine	Enable federated storage engine	no	5.1.3	5.1.9
--with-gnu-ld	Assume the C compiler uses GNU ld	no		

Formats	Description	Default	Introduced	Removed
--with-innodb	Enable innobase storage engine	no	5.1.3	5.1.9
--with-lib-ccflags	Extra CC options for libraries			
--with-libwrap	Compile in libwrap (tcp_wrappers) support			
--with-low-memory	Try to use less memory to compile to avoid memory limitations			
--with-machine-type	Set the machine type, like "powerpc"			
--with-maria-temp-tables	Make the temporary tables within MySQL use the Maria storage engine		5.1.24	
--with-max-indexes	Sets the maximum number of indexes per table	64		
--with-mysqld-ldflags	Extra linking arguments for mysqld			
--with-mysqld-libs	Extra libraries to link with for mysqld			
--with-mysqld-user	What user the mysqld daemon shall be run as			
--with-mysqlmanager	Build the mysqlmanager binary	Build if server is built		
--with-named-curses-libs	Use specified curses libraries			
--with-named-thread-libs	Use specified thread libraries			
--with-ndb-ccflags	Extra CC options for ndb compile			
--with-ndb-docs	Include the NDB Cluster ndbapi and mgmapi documentation			
--with-ndb-port	Port for NDB Cluster management server			
--with-ndb-port-base	Port for NDB Cluster management server			
--with-ndb-sci	Provide MySQL with a custom location of sci library			
--with-ndb-test	Include the NDB Cluster ndbapi test programs			
--with-ndbcluster	Include the NDB Cluster table handler	no		5.1.9
--with-openssl	Include the OpenSSL support			5.1.9

Formats	Description	Default	Introduced	Removed
--with-openssl-includes	Find OpenSSL headers in DIR			5.1.9
--with-openssl-libs	Find OpenSSL libraries in DIR			5.1.9
--with-other-libc	Link against libc and other standard libraries installed in the specified nonstandard location			
--with-pic	Try to use only PIC/non-PIC objects	Use both		
--with-plugin-PLUGIN	Forces the named plugin to be linked into mysqld statically		5.1.11	
--with-plugins	Plugins to include in mysqld	none	5.1.11	
--with-pstack	Use the pstack backtrace library			5.1.54
--with-pthread	Force use of pthread library			
--with-row-based-replication	Include row-based replication		5.1.5	5.1.6
--with-server-suffix	Append value to the version string			
--with-ssl	Include SSL support		5.1.11	
--with-system-type	Set the system type, like "sun-solaris10"			
--with-tags	Include additional configurations	automatic		
--with-tcp-port	Which port to use for MySQL services	3306		
--with-unix-socket-path	Where to put the unix-domain socket			
--with-yassl	Include the yaSSL support			5.1.9
--with-zlib-dir	Provide MySQL with a custom location of compression library			
--without-PACKAGE	Do not use PACKAGE			
--without-bench	Skip building of the benchmark suite			5.1.11
--without-debug	Build a production version without debugging code			5.1.6
--without-docs	Skip building of the documentation			
--without-extra-tools	Skip building utilities in the tools directory			5.1.9
--without-geometry	Do not build geometry-related parts			

Formats	Description	Default	Introduced	Removed
`--without-libedit`	Use system libedit instead of bundled copy			
`--without-man`	Skip building of the man pages			
`--without-ndb-binlog`	Disable ndb binlog		5.1.6	
`--without-ndb-debug`	Disable special ndb debug features			
`--without-plugin-PLUGIN`	Exclude PLUGIN		5.1.11	
`--without-query-cache`	Do not build query cache			
`--without-readline`	Use system readline instead of bundled copy			
`--without-row-based-replication`	Don't include row-based replication		5.1.7	5.1.14
`--without-server`	Only build the client			
`--without-uca`	Skip building of the national Unicode collations			

If you are using a version of `gcc` recent enough to understand the `-fno-exceptions` option, it is *very important* that you use this option. Otherwise, you may compile a binary that crashes randomly. Also use `-felide-constructors` and `-fno-rtti` along with `-fno-exceptions`. When in doubt, do the following:

```
CFLAGS="-O3" CXX=gcc CXXFLAGS="-O3 -felide-constructors \
       -fno-exceptions -fno-rtti" ./configure \
       --prefix=/usr/local/mysql --enable-assembler \
       --with-mysqld-ldflags=-all-static
```

On most systems, this gives you a fast and stable binary.

When compiling from source, you should also be aware of any platform specific considerations that may influence and impact the build process. Knowing and applying this information will help to ensure you get the best performance and most stable binary for your chosen platform. For more information, use the following sections:

- Chapter 8, *Notes on Installing MySQL on AIX from Source*
- Chapter 9, *Notes on Installing MySQL on HP-UX from Source*
- Chapter 7, *Notes on Installing MySQL on Solaris from Source*

Some of the `configure` options available are described here. For options that may be of use if you have difficulties building MySQL, see Chapter 5, *Dealing with Problems Compiling MySQL*.

Many options configure compile-time defaults that can be overridden at server startup. For example, the `--prefix`, `--with-tcp-port`, and `with-unix-socket-path` options that configure the default installation base directory location, TCP/IP port number, and Unix socket file can be changed at server startup with the `--basedir`, `--port`, and `--socket` options for `mysqld`.

- To compile just the MySQL client libraries and client programs and not the server, use the `--without-server` option:

```
shell> ./configure --without-server
```

If you have no C++ compiler, some client programs such as `mysql` cannot be compiled because they require C++. In this case, you can remove the code in `configure` that tests for the C++ compiler and then run `./configure` with the `--without-server` option. The compile step should still try to build all clients, but you can ignore any warnings about files such as `mysql.cc`. (If `make` stops, try `make -k` to tell it to continue with the rest of the build even if errors occur.)

- To build the embedded MySQL library (`libmysqld.a`), use the `--with-embedded-server` option.

- To place your log files and database directories elsewhere than under `/usr/local/var`, use a `configure` command something like one of these:

```
shell> ./configure --prefix=/usr/local/mysql
shell> ./configure --prefix=/usr/local \
          --localstatedir=/usr/local/mysql/data
```

The first command changes the installation prefix so that everything is installed under `/usr/local/mysql` rather than the default of `/usr/local`. The second command preserves the default installation prefix, but overrides the default location for database directories (normally `/usr/local/var`) and changes it to `/usr/local/mysql/data`.

You can also specify the installation directory and data directory locations at server startup time by using the `--basedir` and `--datadir` options. These can be given on the command line or in an MySQL option file, although it is more common to use an option file. See Using Option Files.

- The `--with-tcp-port` option specifies the port number on which the server listens for TCP/IP connections. The default is port 3306. To listen on a different port, use a `configure` command like this:

```
shell> ./configure --with-tcp-port=3307
```

- On Unix, if you want the MySQL socket file location to be somewhere other than the default location (normally in the directory `/tmp` or `/var/run`), use a `configure` command like this:

```
shell> ./configure \
          --with-unix-socket-path=/usr/local/mysql/tmp/mysql.sock
```

The socket file name must be an absolute path name. You can also change the location of `mysql.sock` at server startup by using a MySQL option file. See How to Protect or Change the MySQL Unix Socket File.

- To compile statically linked programs (for example, to make a binary distribution, to get better performance, or to work around problems with some Red Hat Linux distributions), run `configure` like this:

```
shell> ./configure --with-client-ldflags=-all-static \
          --with-mysqld-ldflags=-all-static
```

- If you are using `gcc` and do not have `libg++` or `libstdc++` installed, you can tell `configure` to use `gcc` as your C++ compiler:

```
shell> CC=gcc CXX=gcc ./configure
```

When you use `gcc` as your C++ compiler, it does not attempt to link in `libg++` or `libstdc++`. This may be a good thing to do even if you have those libraries installed. Some versions of them have caused strange problems for MySQL users in the past.

In most cases, you can get a reasonably optimized MySQL binary by using the following options on the `configure` line:

```
--prefix=/usr/local/mysql --enable-assembler \
--with-mysqld-ldflags=-all-static
```

The full `configure` line would, in other words, be something like the following for all recent `gcc` versions:

```
CFLAGS="-O3 -mpentiumpro" CXX=gcc CXXFLAGS="-O3 -mpentiumpro \
-felide-constructors -fno-exceptions -fno-rtti" ./configure \
--prefix=/usr/local/mysql --enable-assembler \
--with-mysqld-ldflags=-all-static
```

The binaries we provide on the MySQL Web site at http://dev.mysql.com/downloads/ are all compiled with full optimization and should work well for most users. See Installing MySQL on Unix/Linux Using Generic Binaries.

- If the build fails and produces errors about your compiler or linker not being able to create the shared library `libmysqlclient.so.N` (where `N` is a version number), you can work around this problem by giving the `--disable-shared` option to `configure`. In this case, `configure` does not build a shared `libmysqlclient.so.N` library.

- By default, MySQL uses the `latin1` (cp1252 West European) character set. To change the default set, use the `--with-charset` option:

```
shell> ./configure --with-charset=CHARSET
```

`CHARSET` may be one of `binary`, `armscii8`, `ascii`, `big5`, `cp1250`, `cp1251`, `cp1256`, `cp1257`, `cp850`, `cp852`, `cp866`, `cp932`, `dec8`, `eucjpms`, `euckr`, `gb2312`, `gbk`, `geostd8`, `greek`, `hebrew`, `hp8`, `keybcs2`, `koi8r`, `koi8u`, `latin1`, `latin2`, `latin5`, `latin7`, `macce`, `macroman`, `sjis`, `swe7`, `tis620`, `ucs2`, `ujis`, `utf8`. (Additional character sets might be available. Check the output from `./configure --help` for the current list.)

The default collation may also be specified. MySQL uses the `latin1_swedish_ci` collation by default. To change this, use the `--with-collation` option:

```
shell> ./configure --with-collation=COLLATION
```

To change both the character set and the collation, use both the `--with-charset` and `--with-collation` options. The collation must be a legal collation for the character set. (Use the `SHOW COLLATION` statement to determine which collations are available for each character set.)

With the `configure` option `--with-extra-charsets=LIST`, you can define which additional character sets should be compiled into the server. `LIST` is one of the following:

- A list of character set names separated by spaces
- `complex` to include all character sets that can't be dynamically loaded

- `all` to include all character sets into the binaries

Clients that want to convert characters between the server and the client should use the `SET NAMES` statement. See Connection Character Sets and Collations.

- To configure MySQL with debugging code, use the `--with-debug` option:

```
shell> ./configure --with-debug
```

This causes a safe memory allocator to be included that can find some errors and that provides output about what is happening. See Debugging and Porting MySQL.

As of MySQL 5.1.12, using `--with-debug` to configure MySQL with debugging support enables you to use the `--debug="d,parser_debug"` option when you start the server. This causes the Bison parser that is used to process SQL statements to dump a parser trace to the server's standard error output. Typically, this output is written to the error log.

- To cause the Debug Sync facility to be compiled into the server, use the `--enable-debug-sync` option. This facility is used for testing and debugging. When compiled in, Debug Sync is disabled by default at runtime. To enable it, start `mysqld` with the `--debug-sync-timeout=N` option, where *`N`* is a timeout value greater than 0. (The default value is 0, which disables Debug Sync.) *`N`* becomes the default timeout for individual synchronization points.

  Debug Sync is also compiled in if you configure with the `--with-debug` option (which implies `--enable-debug-sync`), unless you also use the `--disable-debug-sync` option.

  For a description of the Debug Sync facility and how to use synchronization points, see MySQL Internals: Test Synchronization.

  The `--enable-debug-sync` and `--disable-debug-sync` options were added in MySQL 5.1.41.

- If your client programs are using threads, you must compile a thread-safe version of the MySQL client library with the `--enable-thread-safe-client` configure option. This creates a `libmysqlclient_r` library with which you should link your threaded applications. See Writing C API Threaded Client Programs.

- Some features require that the server be built with compression library support, such as the `COMPRESS()` and `UNCOMPRESS()` functions, and compression of the client/server protocol. The `--with-zlib-dir=no|bundled|DIR` option provides control over compression library support. The value `no` explicitly disables compression support. `bundled` causes the `zlib` library bundled in the MySQL sources to be used. A *`DIR`* path name specifies the directory in which to find the compression library sources.

- It is possible to build MySQL with large table support using the `--with-big-tables` option.

  This option causes the variables that store table row counts to be declared as `unsigned long long` rather than `unsigned long`. This enables tables to hold up to approximately 1.844E+19 ($(2^{32})^2$) rows rather than 2^{32} (~4.295E+09) rows. Previously it was necessary to pass `-DBIG_TABLES` to the compiler manually in order to enable this feature.

- Run `configure` with the `--disable-grant-options` option to cause the `--bootstrap`, `--skip-grant-tables`, and `--init-file` options for `mysqld` to be disabled. For Windows, the `configure.js` script recognizes the `DISABLE_GRANT_OPTIONS` flag, which has the same effect. The capability is available as of MySQL 5.1.15.

- This option allows MySQL Community Server features to be enabled. Additional options may be required for individual features, such as `--enable-profiling` to enable statement profiling. This option was added in MySQL 5.1.24. It is enabled by default as of MySQL 5.1.28; to disable it, use `--disable-community-features`.
- When given with `--enable-community-features`, the `--enable-profiling` option enables the statement profiling capability exposed by the `SHOW PROFILE` and `SHOW PROFILES` statements. (See SHOW PROFILES Syntax.) This option was added in MySQL 5.1.24. It is enabled by default as of MySQL 5.1.28; to disable it, use `--disable-profiling`.
- See General Installation Guidance, for options that pertain to particular operating systems.
- See Building MySQL with Support for Secure Connections, for options that pertain to configuring MySQL to support secure (encrypted) connections.
- Several `configure` options apply to plugin selection and building:

```
--with-plugins=PLUGIN[,PLUGIN]...
--with-plugins=GROUP
--with-plugin-PLUGIN
--without-plugin-PLUGIN
```

  `PLUGIN` is an individual plugin name such as `csv` or `archive`.

  As shorthand, *`GROUP`* is a configuration group name such as `none` (select no plugins) or `all` (select all plugins).

  You can build a plugin as static (compiled into the server) or dynamic (built as a dynamic library that must be installed using the `INSTALL PLUGIN` statement or the `--plugin-load` option before it can be used). Some plugins might not support static or dynamic build.

  `configure --help` shows the following information pertaining to plugins:

  - The plugin-related options
  - The names of all available plugins
  - For each plugin, a description of its purpose, which build types it supports (static or dynamic), and which plugin groups it is a part of.

  `--with-plugins` can take a list of one or more plugin names separated by commas, or a plugin group name. The named plugins are configured to be built as static plugins.

  `--with-plugin-`*`PLUGIN`* configures the given plugin to be built as a static plugin.

  `--without-plugin-`*`PLUGIN`* disables the given plugin from being built.

  If a plugin is named both with a `--with` and `--without` option, the result is undefined.

  For any plugin that is not explicitly selected or disabled, it is selected to be built dynamically if it supports dynamic build, and not built if it does not support dynamic build. (Thus, in the case that no plugin options are given, all plugins that support dynamic build are selected to be built as dynamic plugins. Plugins that do not support dynamic build are not built.)

Chapter 5 Dealing with Problems Compiling MySQL

All MySQL programs compile cleanly for us with no warnings on Solaris or Linux using `gcc`. On other systems, warnings may occur due to differences in system include files. For other problems, check the following list.

The solution to many problems involves reconfiguring. If you do need to reconfigure, take note of the following:

- If `configure` is run after it has previously been run, it may use information that was gathered during its previous invocation. This information is stored in `config.cache`. When `configure` starts up, it looks for that file and reads its contents if it exists, on the assumption that the information is still correct. That assumption is invalid when you reconfigure.
- Each time you run `configure`, you must run `make` again to recompile. However, you may want to remove old object files from previous builds first because they were compiled using different configuration options.

To prevent old configuration information or object files from being used, run these commands before re-running `configure`:

```
shell> rm config.cache
shell> make clean
```

Alternatively, you can run `make distclean`.

The following list describes some of the problems that have been found to occur most often when compiling MySQL:

- If you get errors such as the ones shown here when compiling `sql_yacc.cc`, you probably have run out of memory or swap space:

  ```
  Internal compiler error: program cc1plus got fatal signal 11
  Out of virtual memory
  Virtual memory exhausted
  ```

  The problem is that `gcc` requires a huge amount of memory to compile `sql_yacc.cc` with inline functions. Try running `configure` with the `--with-low-memory` option:

  ```
  shell> ./configure --with-low-memory
  ```

  This option causes `-fno-inline` to be added to the compile line if you are using `gcc` and `-O0` if you are using something else. You should try the `--with-low-memory` option even if you have so much memory and swap space that you think you can't possibly have run out. This problem has been observed to occur even on systems with generous hardware configurations, and the `--with-low-memory` option usually fixes it.

- By default, `configure` picks `c++` as the compiler name and GNU `c++` links with `-lg++`. If you are using `gcc`, that behavior can cause problems during configuration such as this:

  ```
  configure: error: installation or configuration problem:
  C++ compiler cannot create executables.
  ```

  You might also observe problems during compilation related to `g++`, `libg++`, or `libstdc++`.

One cause of these problems is that you may not have `g++`, or you may have `g++` but not `libg++`, or `libstdc++`. Take a look at the `config.log` file. It should contain the exact reason why your C++ compiler did not work. To work around these problems, you can use `gcc` as your C++ compiler. Try setting the environment variable `CXX` to `"gcc -O3"`. For example:

```
shell> CXX="gcc -O3" ./configure
```

This works because `gcc` compiles C++ source files as well as `g++` does, but does not link in `libg++` or `libstdc++` by default.

Another way to fix these problems is to install `g++`, `libg++`, and `libstdc++`. However, do not use `libg++` or `libstdc++` with MySQL because this only increases the binary size of `mysqld` without providing any benefits. Some versions of these libraries have also caused strange problems for MySQL users in the past.

- To define flags to be used by your C or C++ compilers, specify them using the `CFLAGS` and `CXXFLAGS` environment variables. You can also specify the compiler names this way using `CC` and `CXX`. For example:

```
shell> CC=gcc
shell> CFLAGS=-O3
shell> CXX=gcc
shell> CXXFLAGS=-O3
shell> export CC CFLAGS CXX CXXFLAGS
```

To see what flags you might need to specify, invoke `mysql_config` with the `--cflags` option.

- If you get errors such as those shown here when compiling `mysqld`, `configure` did not correctly detect the type of the last argument to `accept()`, `getsockname()`, or `getpeername()`:

```
cxx: Error: mysqld.cc, line 645: In this statement, the referenced
     type of the pointer value ''length'' is ''unsigned long'',
     which is not compatible with ''int''.
new_sock = accept(sock, (struct sockaddr *)&cAddr, &length);
```

To fix this, edit the `config.h` file (which is generated by `configure`). Look for these lines:

```
/* Define as the base type of the last arg to accept */
#define SOCKET_SIZE_TYPE XXX
```

Change `XXX` to `size_t` or `int`, depending on your operating system. (You must do this each time you run `configure` because `configure` regenerates `config.h`.)

- If your compile fails with errors such as any of the following, you must upgrade your version of `make` to GNU `make`:

```
make: Fatal error in reader: Makefile, line 18:
Badly formed macro assignment
```

Or:

```
make: file `Makefile' line 18: Must be a separator (:
```

Or:

```
pthread.h: No such file or directory
```

Solaris and FreeBSD are known to have troublesome `make` programs.

GNU `make` 3.75 is known to work.

- The `sql_yacc.cc` file is generated from `sql_yacc.yy`. Normally, the build process does not need to create `sql_yacc.cc` because MySQL comes with a pregenerated copy. However, if you do need to re-create it, you might encounter this error:

```
"sql_yacc.yy", line xxx fatal: default action causes potential...
```

This is a sign that your version of `yacc` is deficient. You probably need to install `bison` (the GNU version of `yacc`) and use that instead.

Versions of `bison` older than 1.75 may report this error:

```
sql_yacc.yy:#####: fatal error: maximum table size (32767) exceeded
```

The maximum table size is not actually exceeded; the error is caused by bugs in older versions of `bison`.

- On Debian Linux 3.0, you need to install `gawk` instead of the default `mawk`.
- If you get a compilation error on Linux (for example, SuSE Linux 8.1 or Red Hat Linux 7.3) similar to the following one, you probably do not have `g++` installed:

```
libmysql.c:1329: warning: passing arg 5 of `gethostbyname_r' from
incompatible pointer type
libmysql.c:1329: too few arguments to function `gethostbyname_r'
libmysql.c:1329: warning: assignment makes pointer from integer
without a cast
make[2]: *** [libmysql.lo] Error 1
```

By default, the `configure` script attempts to determine the correct number of arguments by using `g++` (the GNU C++ compiler). This test yields incorrect results if `g++` is not installed. There are two ways to work around this problem:

- Make sure that the GNU C++ `g++` is installed. On some Linux distributions, the required package is called `gpp`; on others, it is named `gcc-c++`.
- Use `gcc` as your C++ compiler by setting the `CXX` environment variable to `gcc`:

```
export CXX="gcc"
```

You must run `configure` again after making either of those changes.

For information about acquiring or updating tools, see the system requirements in Chapter 1, *Installing MySQL from Source*.

Chapter 6 Installing MySQL from Source on Windows

These instructions describe how to build binaries from source for MySQL 5.1 on Windows. Instructions are provided for building binaries from a standard source distribution or from the Bazaar tree that contains the latest development source.

> **Note**
>
> The instructions here are strictly for users who want to test MySQL on Microsoft Windows from the latest source distribution or from the Bazaar tree. For production use, we do not advise using a MySQL server built by yourself from source. Normally, it is best to use precompiled binary distributions of MySQL that are built specifically for optimal performance on Windows by Oracle Corporation. Instructions for installing binary distributions are available in Installing MySQL on Microsoft Windows.

To build MySQL on Windows from source, you must satisfy the following system, compiler, and resource requirements:

- Windows 2000, Windows XP, or newer version.

  Windows Vista is supported when using Visual Studio 2005 provided you have installed the following updates:

  - Microsoft Visual Studio 2005 Professional Edition - ENU Service Pack 1 (KB926601)
  - Security Update for Microsoft Visual Studio 2005 Professional Edition - ENU (KB937061)
  - Update for Microsoft Visual Studio 2005 Professional Edition - ENU (KB932232)

- `CMake`, which can be downloaded from http://www.cmake.org. After installing, modify your `PATH` environment variable to include the directory where `cmake` is located.
- Microsoft Visual C++ 2005 Express Edition, Visual Studio .Net 2003 (7.1), or Visual Studio 2005 (8.0) compiler system.
- If you are using Visual C++ 2005 Express Edition, you must also install an appropriate Platform SDK. More information and links to downloads for various Windows platforms is available from http://www.microsoft.com/downloads/details.aspx?familyid=0baf2b35-c656-4969-ace8-e4c0c0716adb.
- If you are compiling from a Bazaar tree or making changes to the parser, you need `bison` for Windows, which can be downloaded from http://gnuwin32.sourceforge.net/packages/bison.htm. Download the package labeled “Complete package, excluding sources”. After installing the package, modify your `PATH` environment variable to include the directory where `bison` is located.

  > **Note**
  >
  > On Windows, the default location for `bison` is the `C:\Program Files\GnuWin32` directory. Some utilities, including `m4`, may fail to find `bison` because of the space in the directory name. You can resolve this by installing into a directory that does not contain a space; for example `C:\GnuWin32`.

- Cygwin might be necessary if you want to run the test script or package the compiled binaries and support files into a Zip archive. (Cygwin is needed only to test or package the distribution, not to build it.) Cygwin is available from http://cygwin.com.
- 3GB to 5GB of disk space.

You also need a MySQL source distribution for Windows, which can be obtained two ways:

- Obtain a source distribution packaged by Oracle Corporation. These are available from http://dev.mysql.com/downloads/.
- Package a source distribution yourself from the latest Bazaar developer source tree. For instructions on pulling the latest source files, see Chapter 3, *Installing MySQL Using a Development Source Tree*.

If you find something not working as expected, or you have suggestions about ways to improve the current build process on Windows, please send a message to the `win32` mailing list. See MySQL Mailing Lists.

> **Note**
>
> To compile from the source code on Windows you must use the standard source distribution (for example, `mysql-5.1.73.zip`) or `mysql-5.1.73.tar.gz`). You build from the same distribution as used to build MySQL on Unix, Linux and other platforms. Do *not* use the Windows Source distributions as they do not contain the necessary configuration script and other files.

Follow this procedure to build MySQL:

1. If you are installing from a packaged source distribution, create a work directory (for example, `C:\workdir`), and unpack the source distribution there using `WinZip` or another Windows tool that can read `.zip` files. This directory is the work directory in the following instructions.

   > **Note**
   >
   > Commands that are located in the `win` directory should be executed from the top-level source directory. Do not change location into the `win` directory, as the commands will not execute correctly.

2. Start a command shell. If you have not configured the `PATH` and other environment variables for all command shells, you may be able to start a command shell from the **Start Menu** within the Windows Visual Studio menu that contains the necessary environment changes.

3. Within the command shell, navigate to the work directory and run the following command:

   ```
   C:\workdir>win\configure.js options
   ```

   If you have associated the `.js` file extension with an application such as a text editor, then you may need to use the following command to force `configure.js` to be executed as a script:

   ```
   C:\workdir>cscript win\configure.js options
   ```

   These options are available for `configure.js`:

   - `WITH_INNOBASE_STORAGE_ENGINE`: Enable the `InnoDB` storage engine.
   - `WITH_PARTITION_STORAGE_ENGINE`: Enable user-defined partitioning.
   - `WITH_ARCHIVE_STORAGE_ENGINE`: Enable the `ARCHIVE` storage engine.
   - `WITH_BLACKHOLE_STORAGE_ENGINE`: Enable the `BLACKHOLE` storage engine.
   - `WITH_EXAMPLE_STORAGE_ENGINE`: Enable the `EXAMPLE` storage engine.
   - `WITH_FEDERATED_STORAGE_ENGINE`: Enable the `FEDERATED` storage engine.

- `WITH_NDBCLUSTER_STORAGE_ENGINE`: Enable the `NDBCLUSTER` storage engine in the MySQL server; cause binaries for the MySQL Cluster management and data node, management client, and other programs to be built.

  This option is supported only in MySQL Cluster NDB 7.0 and later (`NDBCLUSTER` storage engine versions 6.4.0 and later) using the MySQL Cluster sources. It cannot be used to enable clustering support in other MySQL source trees or distributions.

- `MYSQL_SERVER_SUFFIX=`*`suffix`*: Server suffix, default none.

- `COMPILATION_COMMENT=`*`comment`*: Server comment, default "Source distribution".

- `MYSQL_TCP_PORT=`*`port`*: Server port, default 3306.

- `DISABLE_GRANT_OPTIONS`: Disables the `--bootstrap`, `--skip-grant-tables`, and `--init-file` options for `mysqld`. This option is available as of MySQL 5.1.15.

For example (type the command on one line):

```
C:\workdir>win\configure.js WITH_INNOBASE_STORAGE_ENGINE
            WITH_PARTITION_STORAGE_ENGINE MYSQL_SERVER_SUFFIX=-pro
```

4. From the work directory, execute the `win\build-vs9.bat` (Windows Visual Studio 2008), `win\build-vs8.bat` (Windows Visual Studio 2005), or `win\build-vs71.bat` (Windows Visual Studio 2003) script, depending on the version of Visual Studio you have installed. The script invokes `CMake`, which generates the `mysql.sln` solution file.

   You can also use the corresponding 64-bit file (for example `win\build-vs8_x64.bat` or `win\build-vs9_x64.bat`) to build the 64-bit version of MySQL. However, you cannot build the 64-bit version with Visual Studio Express Edition. You must use Visual Studio 2005 (8.0) or higher.

5. From the work directory, open the generated `mysql.sln` file with Visual Studio and select the proper configuration using the **Configuration** menu. The menu provides **Debug**, **Release**, **RelwithDebInfo**, **MinRelInfo** options. Then select **Solution** > **Build** to build the solution.

   Remember the configuration that you use in this step. It is important later when you run the test script because that script needs to know which configuration you used.

6. Test the server. The server built using the preceding instructions expects that the MySQL base directory and data directory are `C:\mysql` and `C:\mysql\data` by default. If you want to test your server using the source tree root directory and its data directory as the base directory and data directory, you need to tell the server their path names. You can either do this on the command line with the `--basedir` and `--datadir` options, or by placing appropriate options in an option file. (See Using Option Files.) If you have an existing data directory elsewhere that you want to use, you can specify its path name instead.

   When the server is running in standalone fashion or as a service based on your configuration, try to connect to it from the `mysql` interactive command-line utility.

   You can also run the standard test script, `mysql-test-run.pl`. This script is written in Perl, so you'll need either Cygwin or ActiveState Perl to run it. You may also need to install the modules required by the script. To run the test script, change location into the `mysql-test` directory under the work directory, set the `MTR_VS_CONFIG` environment variable to the configuration you selected earlier (or use the `--vs-config` option), and invoke `mysql-test-run.pl`. For example (using Cygwin and the `bash` shell):

```
shell> cd mysql-test
shell> export MTR_VS_CONFIG=debug
shell> ./mysql-test-run.pl --force --timer
shell> ./mysql-test-run.pl --force --timer --ps-protocol
```

When you are satisfied that the programs you have built are working correctly, stop the server. Now you can install the distribution. One way to do this is to use the `make_win_bin_dist` script in the `scripts` directory of the MySQL source distribution (see `make_win_bin_dist` — Package MySQL Distribution as Zip Archive). This is a shell script, so you must have Cygwin installed if you want to use it. It creates a Zip archive of the built executables and support files that you can unpack in the location at which you want to install MySQL.

It is also possible to install MySQL by copying directories and files directly:

1. Create the directories where you want to install MySQL. For example, to install into `C:\mysql`, use these commands:

   ```
   C:\> mkdir C:\mysql
   C:\> mkdir C:\mysql\bin
   C:\> mkdir C:\mysql\data
   C:\> mkdir C:\mysql\share
   C:\> mkdir C:\mysql\scripts
   ```

   If you want to compile other clients and link them to MySQL, you should also create several additional directories:

   ```
   C:\> mkdir C:\mysql\include
   C:\> mkdir C:\mysql\lib
   C:\> mkdir C:\mysql\lib\debug
   C:\> mkdir C:\mysql\lib\opt
   ```

   If you want to benchmark MySQL, create this directory:

   ```
   C:\> mkdir C:\mysql\sql-bench
   ```

   Benchmarking requires Perl support for MySQL. See Perl Installation Notes.

2. From the work directory, copy into the `C:\mysql` directory the following files and directories:

   ```
   C:\> cd \workdir
   C:\workdir> mkdir C:\mysql
   C:\workdir> mkdir C:\mysql\bin
   C:\workdir> copy client\Release\*.exe C:\mysql\bin
   C:\workdir> copy sql\Release\mysqld.exe C:\mysql\bin\mysqld.exe
   C:\workdir> xcopy scripts\*.* C:\mysql\scripts /E
   C:\workdir> xcopy share\*.* C:\mysql\share /E
   ```

   If you want to compile other clients and link them to MySQL, you should also copy several libraries and header files:

   ```
   C:\workdir> copy lib\Release\mysqlclient.lib C:\mysql\lib\debug
   C:\workdir> copy lib\Release\libmysql.* C:\mysql\lib\debug
   C:\workdir> copy lib\Release\zlib.* C:\mysql\lib\debug
   C:\workdir> copy lib\Release\mysqlclient.lib C:\mysql\lib\opt
   C:\workdir> copy lib\Release\libmysql.* C:\mysql\lib\opt
   C:\workdir> copy lib\Release\zlib.* C:\mysql\lib\opt
   ```

```
C:\workdir> copy include\*.h C:\mysql\include
C:\workdir> copy libmysql\libmysql.def C:\mysql\include
```

> **Note**
>
> If you have compiled a Debug solution, rather than a Release solution, install it by replacing `Release` with `Debug` in the source file names just shown.

If you want to benchmark MySQL, you should also do this:

```
C:\workdir> xcopy sql-bench\*.* C:\mysql\bench /E
```

After installation, set up and start the server in the same way as for binary Windows distributions. This includes creating the system tables by running `mysql_install_db`. For more information, see Installing MySQL on Microsoft Windows.

Chapter 7 Notes on Installing MySQL on Solaris from Source

When building MySQL on Solaris you can use either the Sun Studio or GNU cc compilers. For more information on specific notes and environments, use the following hints.

- When building you should ensure that your `PATH` variable includes the necessary tools, including `ar` for building libraries. Some tools are located in `/usr/ccs/bin`.
- When running `configure`, you should specify the C and C++ compiler explicitly to ensure that the right C compiler combination is used:

```
CC=gcc CXX=g++ ./configure
```

- If you have an UltraSPARC system, you can get 4% better performance by adding `-mcpu=v8 -Wa,-xarch=v8plusa` to the `CFLAGS` and `CXXFLAGS` environment variables.
- If you have Sun's Forte 5.00 (or newer) or Sun Studio compiler, you can run `configure` like this:

```
CC=cc CFLAGS="-Xa -fast -native -xstrconst -mt" \
CXX=CC CXXFLAGS="-noex -mt" \
./configure --prefix=/usr/local/mysql --enable-assembler
```

- To create a 64-bit SPARC binary with Sun's Forte or Sun Studio compiler, use the following configuration options:

```
CC=cc CFLAGS="-Xa -fast -native -xstrconst -mt -xarch=v9" \
CXX=CC CXXFLAGS="-noex -mt -xarch=v9" ASFLAGS="-xarch=v9" \
./configure --prefix=/usr/local/mysql --enable-assembler
```

  To create a 64-bit Solaris binary using `gcc`, add `-m64` to `CFLAGS` and `CXXFLAGS` and remove `--enable-assembler` from the `configure` line.

  In the MySQL benchmarks, we obtained a 4% speed increase on UltraSPARC when using Forte 5.0 in 32-bit mode, as compared to using `gcc` 3.2 with the `-mcpu` flag.

  If you create a 64-bit `mysqld` binary, it is 4% slower than the 32-bit binary, but can handle more threads and memory.

- If you get a problem with `fdatasync` or `sched_yield`, you can fix this by adding `LIBS=-lrt` to the `configure` line
- Solaris does not provide static versions of all system libraries (`libpthreads` and `libdl`), so you cannot compile MySQL with `--static`. If you try to do so, you get one of the following errors:

```
ld: fatal: library -ldl: not found
undefined reference to `dlopen'
cannot find -lrt
```

- If you link your own MySQL client programs, you may see the following error at runtime:

```
ld.so.1: fatal: libmysqlclient.so.#:
open failed: No such file or directory
```

  To avoid this problem, use one of the following methods:

- Use the `crle` tool to add the directory containing the `libmysqlclient` library file to the list of standard library directories. You need administrator privileges to do this. Make sure you update the library information, rather than replace it with the new path. For example, the following command adds the directory to the list of standard directories searched for libraries.

  ```
  crle -u -l /usr/local/mysql/lib
  ```

  For 64-bit libraries, add the `-64` option:

  ```
  crle -64 -u -l /usr/local/mysql/lib
  ```

- Link clients with the `-Wl,r/full/path/to/libmysqlclient.so` flag rather than with `-Lpath`).
- Copy `libmysqlclient.so` to `/usr/lib`.
- Add the path name of the directory where `libmysqlclient.so` is located to the `LD_RUN_PATH` environment variable before running your client.

- If you have problems with `configure` trying to link with `-lz` when you do not have `zlib` installed, you have two options:
  - If you want to be able to use the compressed communication protocol, obtain and install `zlib` from `ftp.gnu.org`.
  - To build without `zlib`, run `configure` with the `--with-named-z-libs=no` option when building MySQL.
- If you are using `gcc` and have problems with loading user-defined functions (UDFs) into MySQL, try adding `-lgcc` to the link line for the UDF.

Chapter 8 Notes on Installing MySQL on AIX from Source

General notes on building MySQL from source on IBM AIX:

- Automatic `xlC` detection is missing from Autoconf, so a number of variables need to be set before running `configure`. The following example uses the IBM compiler:

```
export CC="xlc_r -ma -O3 -qstrict -qoptimize=3 -qmaxmem=8192 "
export CXX="xlC_r -ma -O3 -qstrict -qoptimize=3 -qmaxmem=8192"
export CFLAGS="-I /usr/local/include"
export LDFLAGS="-L /usr/local/lib"
export CPPFLAGS=$CFLAGS
export CXXFLAGS=$CFLAGS
./configure --prefix=/usr/local \
                --localstatedir=/var/mysql \
                --sbindir='/usr/local/bin' \
                --libexecdir='/usr/local/bin' \
                --enable-thread-safe-client \
                --enable-large-files
```

  The preceding options are used to compile the MySQL distribution that can be found at http://www-frec.bull.com/.

- If you change the `-O3` to `-O2` in the preceding `configure` line, you must also remove the `-qstrict` option. This is a limitation in the IBM C compiler.

- If you compile MySQL with `gcc`, you *must* use the `-fno-exceptions` flag because the exception handling in `gcc` is not thread-safe. There are also some known problems with IBM's assembler that may cause it to generate bad code when used with `gcc`.

- If you have problems with signals (MySQL dies unexpectedly under high load), you may have found an OS bug with threads and signals. In this case, you can tell MySQL not to use signals by configuring as follows:

```
CFLAGS=-DDONT_USE_THR_ALARM CXX=gcc \
CXXFLAGS="-felide-constructors -fno-exceptions -fno-rtti \
-DDONT_USE_THR_ALARM" \
./configure --prefix=/usr/local/mysql --with-debug \
    --with-low-memory
```

  This does not affect the performance of MySQL, but has the side effect that you cannot kill clients that are “sleeping” on a connection with `mysqladmin kill` or `mysqladmin shutdown`. Instead, the client dies when it issues its next command.

- On some versions of AIX, linking with `libbind.a` makes `getservbyname()` dump core. This is an AIX bug and should be reported to IBM.

Chapter 9 Notes on Installing MySQL on HP-UX from Source

General notes on compiling MySQL on HP-UX.

- If you are using HP-UX compiler, you can use the following command (which has been tested with `cc` B.11.11.04):

  ```
  CC=cc CXX=aCC CFLAGS=+DD64 CXXFLAGS=+DD64 ./configure \
      --with-extra-character-set=complex
  ```

  You can ignore any errors of the following type:

  ```
  aCC: warning 901: unknown option: `-3': use +help for online
  documentation
  ```

- If you get the following error from `configure`, verify that you do not have the path to the K&R compiler before the path to the HP-UX C and C++ compiler:

  ```
  checking for cc option to accept ANSI C... no
  configure: error: MySQL requires an ANSI C compiler (and a C++ compiler).
  Try gcc. See the Installation chapter in the Reference Manual.
  ```

- Another reason for compile failure is that you did not define the `+DD64` flags as just described.