Executing Programs from the Command Line

Paul DuBois
paul@kitebird.com

Document revision: 2.0
Last update: 2014-07-29

Note: This is a *MySQL Cookbook* auxiliary document. Much of its content originally was an appendix in the book, but as of the third edition is available separately. There are several references here to the `recipes` software distribution. To obtain this distribution, which accompanies *MySQL Cookbook*, visit the companion web site:

```
http://www.kitebird.com/mysql-cookbook/
```

As you work through *MySQL Cookbook*, you run many examples using the *mysql* client program, and there are lots of programs in the `recipes` distribution for you to try as you read. And of course one purpose of the book is to enable you to write your own MySQL-based programs. Consequently, you must often execute programs at the command line—that is, at the prompt of your shell or command interpreter. For best use of this book, you should also be able to run *mysql* easily (by entering just its name), and you should be able to execute programs from the `recipes` distribution or that you write yourself. To accomplish those goals, it's important that your `PATH` environment variable be set correctly, and that you know how to make programs executable. The discussion here shows how to do those things.

Setting Environment Variables

An environment variable has a value that can be accessed by programs as they run. By setting environment variables, you modify your operating environment. One such variable is `PATH`, which is used by your command interpreter to determine which directories to search when it looks for programs such as *mysql* that you tell it to execute. If your `PATH` is set correctly, you can invoke programs easily no matter what your current directory is. If `PATH` is not set correctly, your command interpreter will not find them. For example, if the `PATH` value does not include the directory where *mysql* is installed, a "command not found" error may occur if you attempt to run *mysql* by entering its name. You must then run it either by specifying its full pathname, or by changing location into the directory where it is installed. Both strategies are inconvenient, and more so with repetition. It's much better to set your `PATH` value to name the directories containing the programs you want to use.

Other environment variables are important in other contexts. For example, to run Perl or Ruby scripts that use module files that you've installed, you may need to set the `PERL5LIB` or `RUBYLIB` variable to tell Perl or Ruby where to find those modules. For Java, the `JAVA_HOME` variable should be set to indicate the location of your Java installation, and `CLASSPATH` should be set to the list of libraries and class files that your Java programs need.

The following discussion describes how to set environment variables. The examples demonstrate how to make it easier to run MySQL programs by modifying your `PATH` setting to add the directory where MySQL programs are installed. However, the discussion applies to other environment variables as well because the variable-setting syntax is the same.

Here are some general principles to keep in mind regarding environment variables:

- It's possible to set an environment variable from the command line, but the variable will retain its value only until you exit the command interpreter. To cause the setting to take effect permanently, set it in the appropriate shell startup file on Unix or by using the Environment Variables interface on Windows (this can be found on the System Control Panel under Advanced). The change takes effect for subsequent invocations of your command interpreter (for example, your next login on Unix, or the next console window that you open on Windows).
- Some variables (PATH among them) have a value that lists the pathnames for one or more directories. Under Unix, it's conventional to separate the pathnames using the colon character (`:`). Under Windows, pathnames can contain colons, so the separator is the semicolon character (`;`).
- To check your current environment variable settings, execute an *env* or *printenv* command on Unix, or a *set* command in a console window on Windows.

Setting the PATH Variable on Unix

Your choice of command interpreter determines the syntax for setting environment variables. In addition, for settings that you put in a startup file, the file to use is interpreter-specific. The following table shows typical startup files for commonly used Unix command interpreters. If you've never looked through your command interpreter's startup files, it's a good idea to do so to familiarize yourself with their contents.

Command interpreter	**Possible startup files**
sh, *bash*, *ksh*	*.profile*, *.bash_profile*, *.bash_login*, *.bashrc*
csh, *tcsh*	*.login*, *.cshrc*, *.tcshrc*

The examples assume that MySQL programs are installed in the */usr/local/mysql/bin* directory; adjust the instructions for the pathname actually used on your system. The examples also assume that you want to add that *bin* directory pathname to an existing PATH value. If you have no PATH setting currently, add the appropriate line or lines to one of the startup files.

- For a shell in the Bourne shell family (*sh*, *bash*, *ksh*), look in your startup files for a line that sets and exports the PATH variable:

  ```
  export PATH=/bin:/usr/bin:/usr/local/bin
  ```

  In that case, change the setting to add the appropriate directory:

  ```
  export PATH=/usr/local/mysql/bin:/bin:/usr/bin:/usr/local/bin
  ```

  The assignment and the export might be on separate lines:

  ```
  PATH=/bin:/usr/bin:/usr/local/bin
  export PATH
  ```

  Change the setting to this:

  ```
  PATH=/usr/local/mysql/bin:/bin:/usr/bin:/usr/local/bin
  export PATH
  ```

- For a shell in the C shell family (*csh*, *tcsh*), look in your startup files for a *setenv PATH* command:

  ```
  setenv PATH /bin:/usr/bin:/usr/local/bin
  ```

  Change the setting to add the appropriate directory:

  ```
  setenv PATH /usr/local/mysql/bin:/bin:/usr/bin:/usr/local/bin
  ```

  It's also possible that your path will be set with a *set path* command, which uses different syntax:

  ```
  set path = (/usr/local/mysql/bin /bin /usr/bin /usr/local/bin)
  ```

  `path` is not an environment variable, but for C shells, setting `path` also sets PATH and vice versa.

Setting the PATH Variable on Windows

To set environment variables on Windows, right-click My Computer on the desktop, then select → Properties → Advanced → Environment Variables. You should see a window that enables you to define environment variables or change the values of existing variables. For example, your PATH variable might have a value like this:

```
C:\WINDOWS;C:\WINDOWS\COMMAND
```

To make it easier to run MySQL programs such as *mysql*, change the value to include the directory where those programs are installed. Adding a directory of *C:\Program Files\MySQL\MySQL Server 5.6\bin* results in a PATH setting that should look like this:

```
C:\Program Files\MySQL\MySQL Server 5.6\bin;C:\WINDOWS;C:\WINDOWS\COMMAND
```

Executing Programs

This section describes how to execute programs from the command line. You can use this information to run programs that you obtain from the `recipes` distribution or that you write yourself. The first set of instructions applies to scripts written in Perl, Ruby, PHP, or Python. A second set of instructions applies to Java programs.

The example programs can be found in the *progdemo* directory of the `recipes` distribution.

Executing Perl, Ruby, PHP, or Python Scripts

The following discussion shows how to execute scripts, using Perl for the examples. The principles are similar for Ruby, PHP, or Python scripts.

Begin with an example script named *perldemo.pl* that consists of a simple print statement:

```
print "I am a Perl program.\n";
```

A script-execution method that works on any platform is to invoke the *perl* program and tell it the name of the script to run:

```
% perl perldemo.pl
I am a Perl program.
```

For a script written in another language, invoke the *ruby*, *php*, or *python* program.

It's also possible to make a script directly executable. The procedure differs for Unix and Windows. Both procedures are described here.

On Unix, to make a script directly executable, include a line at the top of the file that begins with `#!` and that specifies the pathname of the program that should execute the script. Here is a script named *perldemo2.pl* with a `#!` line that names the *perl* program (if *perl* is not located at */usr/bin/perl* on your system, change the pathname):

```
#!/usr/bin/perl
print "I am a Perl program.\n";
```

Next, enable the executable access mode with *chmod +x*:

```
% chmod +x perldemo2.pl
```

At this point, the script can be run the same way as its earlier counterpart (that is, you can use *perl perldemo2.pl*), but now you should also be able to execute it by entering just its name. However, assuming that the script is located in your current directory, your shell might not find it. The shell searches for

programs in the directories named in your PATH environment variable, but for security reasons, the search path for Unix shells often is deliberately set not to include the current directory (`.`). In that case, include a leading path of `./` to explicitly indicate the script's location:

```
% ./perldemo2.pl
I am a Perl program.
```

If you install the script by copying it to a directory that is named in your PATH setting, the leading `./` path is unnecessary when you invoke the script.

On Windows, the procedure for making scripts executable is somewhat different. *chmod* is not used. Instead, programs are treated as executable based on their filename extensions (such as *.exe* or *.bat*). Windows enables you to set up filename mappings that associate a programs with filenames that end with a particular suffix. This means that you can set up associations so that Perl, Ruby, PHP, or Python are used to execute files with names that end in *.pl*, *.rb*, *.php*, or *.py*, respectively. (The installer for a given language might even create the association for you.) Then you can invoke a script with a given extension directly from the command line without naming its language interpreter:

```
C:\> perldemo.pl
I am a Perl program.
```

No leading path is needed to invoke a script that is located in the current directory because the Windows command interpreter includes that directory in its search path by default.

Keep the preceding principles in mind when you run scripts from within a directory of the `recipes` distribution. For example, if your current location is the *metadata* directory of the distribution and you want to execute the *get_server_version.rb* Ruby script, you can do so with either of these commands on Unix:

```
% ruby get_server_version.rb
% ./get_server_version.rb
```

On Windows, you can use either of these commands:

```
C:\> ruby get_server_version.rb
C:\> get_server_version.rb
```

Compiling and Executing Java Programs

Java programming requires a Java Development Kit (JDK). If a JDK is not already installed on your system, obtain one and install it. OS X comes with a JDK. For Solaris, Linux, and Windows, JDKs are available at `http://www.oracle.com/technetwork/java/index.html`.

After verifying that a JDK is installed, set the JAVA_HOME environment variable if it's not already set. Its value should be the pathname of the directory where the JDK is installed. If the JDK installation directory is */usr/local/java/jdk* on Unix or *C:\jdk* on Windows, set JAVA_HOME as follows:

- For a shell in the Bourne shell family (*sh*, *bash*, *ksh*):

  ```
  export JAVA_HOME=/usr/local/java/jdk
  ```

  For the original Bourne shell, *sh*, you may need to split this into two commands:

  ```
  JAVA_HOME=/usr/local/java/jdk
  export JAVA_HOME
  ```

- For a shell in the C shell family (*csh*, *tcsh*):

  ```
  setenv JAVA_HOME=/usr/local/java/jdk
  ```

- For Windows, get to the Environment Variables window as described previously in ''Setting Environment Variables,'' then set JAVA_HOME to this value:

```
C:\jdk
```

Adjust the instructions for the pathname actually used on your system.

With a JDK in place and JAVA_HOME set, you should be able to compile and run Java programs. Here is a sample program:

```
public class JavaDemo
{
  public static void main (String[] args)
  {
    System.out.println ("I am a Java program.");
  }
}
```

The `class` statement indicates the program's name, which in this case is `JavaDemo`. The name of the file containing the program should match this name and include a *.java* extension, so the filename for the program is *JavaDemo.java*. Compile the program using *javac*:

```
% javac JavaDemo.java
```

If you prefer a different Java compiler, just substitute its name. For example, if you'd rather use Jikes, compile the file like this instead:

```
% jikes JavaDemo.java
```

The Java compiler generates compiled byte code to produce a class file named *JavaDemo.class*. Use the *java* program to run the class file (specified without the *.class* extension):

```
% java JavaDemo
I am a Java program.
```

To compile and run Java programs that use MySQL, you need the MySQL Connector/J JDBC driver. If Java cannot find the driver, you must set your CLASSPATH environment variable. Its value should include at least your current directory (`.`) and the pathname of the Connector/J driver. If the driver has a pathname of */usr/local/lib/java/lib/mysql-connector-java-bin.jar* (Unix) or *C:\Java\lib\mysql-connector-java-bin.jar* (Windows), set CLASSPATH as follows:

- For a shell in the Bourne shell family (*sh*, *bash*, *ksh*):

  ```
  export CLASSPATH=.:/usr/local/lib/java/lib/mysql-connector-java-bin.jar
  ```

- For a shell in the C shell family (*csh*, *tcsh*):

  ```
  setenv CLASSPATH .:/usr/local/lib/java/lib/mysql-connector-java-bin.jar
  ```

- For Windows, get to the Environment Variables window as described earlier in ''Setting Environment Variables,'' then set CLASSPATH to this value:

  ```
  .;C:\Java\lib\mysql-connector-java-bin.jar
  ```

Adjust the instructions for the pathname actually used on your system. You might also need to add other class directories or libraries to your CLASSPATH setting. The specifics depend on how your system is set up.

Revision History

- 1.0—The version in *MySQL Cookbook*, second edition, Appendix B.
- 2.0—Initial standalone version, to accompany *MySQL Cookbook*, third edition.