sqlcmd utility

Article • 06/12/2024 • 8 contributors

Feedback

In this article

Applies to: ✓ SQL Server ✓ Azure SQL Database ✓ Azure SQL Managed Instance ✓ Azure Synapse Analytics ✓ Analytics Platform System (PDW)

The **sqlcmd** utility lets you enter Transact-SQL statements, system procedures, and script files through various modes:

- At the command prompt.
- In **Query Editor** in SQLCMD mode.
- In a Windows script file.
- In an operating system (`cmd.exe`) job step of a SQL Server Agent job.

ⓘ **Note**

While Microsoft Entra ID is the new name for Azure Active Directory (Azure AD), to prevent disrupting existing environments, Azure AD still remains in some hardcoded elements such as UI fields, connection providers, error codes, and cmdlets. In this article, the two names are interchangeable.

Find out which version you have installed

There are two versions of **sqlcmd**:

- The `go-mssqldb`-based **sqlcmd**, sometimes styled as **go-sqlcmd**. This version is a standalone tool you can download independently of SQL Server.

- The ODBC-based **sqlcmd**, available with SQL Server or the Microsoft Command Line Utilities, and part of the `mssql-tools` package on Linux.

To determine the version you have installed, run the following statement at the command line:

```
sqlcmd "-?"
```

```
sqlcmd "-?"
sqlcmd -?
```

sqlcmd (Go) | sqlcmd (ODBC)

If you're using the new version of **sqlcmd** (Go), the output is similar to the following example:

```
Version: 1.3.1
```

Check version

You can use `sqlcmd --version` to determine which version is installed. You should have at least version 1.0.0 installed.

Important

Installing **sqlcmd** (Go) via a package manager will replace **sqlcmd** (ODBC) with **sqlcmd** (Go) in your environment path. Any current command line sessions will need to be closed and reopened for this take to effect. **sqlcmd** (ODBC) won't be removed and can still be used by specifying the full path to the executable. You can also update your `PATH` variable to indicate which will take precedence. To do so in Windows 11, open **System settings** and go to **About** > **Advanced system settings**. When **System Properties** opens, select the **Environment Variables** button. In the lower half, under **System variables**, select **Path** and then select **Edit**. If the location **sqlcmd** (Go) is saved to (`C:\Program Files\sqlcmd` is default) is listed before `C:\Program Files\Microsoft SQL Server\<version>\Tools\Binn`, then **sqlcmd** (Go) is used. You can reverse the order to make **sqlcmd** (ODBC) the default again.

Download and install sqlcmd

sqlcmd (Go) sqlcmd (ODBC)

sqlcmd (Go) can be installed cross-platform, on Microsoft Windows, macOS, and Linux. Versions newer than 1.6 might not be available in all package managers. There's no estimated date yet for their availability.

Windows macOS Linux

winget (Windows Package Manager CLI)

1. Install the Windows Package Manager Client if you don't already have it.
2. Run the following command to install **sqlcmd** (Go).

```
winget install sqlcmd
```

Chocolatey

1. Install Chocolatey if you don't already have it.
2. Run the following command to install **sqlcmd** (Go).

```
choco install sqlcmd
```

Direct download

1. Download the corresponding `-windows-amd64.zip` or `-windows-arm.zip` asset from the latest release of **sqlcmd** (Go) from the GitHub code repository.
2. Extract the `sqlcmd.exe` file from the downloaded zip folder.

Preinstalled

Azure Cloud Shell

You can try the **sqlcmd** utility from Azure Cloud Shell, as it's preinstalled by default:

- Launch Cloud Shell

Azure Data Studio

To run SQLCMD statements in Azure Data Studio, select **Enable SQLCMD** from the editor toolbar.

SQL Server Management Studio (SSMS)

To run SQLCMD statements in SQL Server Management Studio (SSMS), select SQLCMD Mode from the top navigation Query Menu dropdown list.

SSMS uses the Microsoft .NET Framework `SqlClient` for execution in regular and SQLCMD mode in **Query Editor**. When **sqlcmd** is run from the command-line, **sqlcmd** uses the ODBC driver. Because different default options may apply, you might see different behavior when you execute the same query in SSMS in SQLCMD Mode and in the **sqlcmd** utility.

Syntax

sqlcmd (Go) sqlcmd (ODBC)

```
Usage:
  sqlcmd [flags]
  sqlcmd [command]

Examples:
# Install/Create, Query, Uninstall SQL Server
  sqlcmd create mssql --accept-eula --using https://aka.
  sqlcmd open ads
  sqlcmd query "SELECT @@version"
  sqlcmd delete
# View configuration information and connection strings
  sqlcmd config view
  sqlcmd config cs

Available Commands:
  completion  Generate the autocompletion script for the
  config      Modify sqlconfig files using subcommands l
  create      Install/Create SQL Server, Azure SQL, and
  delete      Uninstall/Delete the current context
  help        Help about any command
  open        Open tools (e.g ADS) for current context
  query       Run a query against the current context
  start       Start current context
  stop        Stop current context

Flags:
  -?, --?                    help for backwards compatibil
  -h, --help                 help for sqlcmd
      --sqlconfig string     configuration file (default '
      --verbosity int        log level, error=0, warn=1, i
      --version              print version of sqlcmd

Use "sqlcmd [command] --help" for more information about
```

For more in-depth information on **sqlcmd** syntax and use, see ODBC sqlcmd syntax.

Breaking changes from sqlcmd (ODBC)

Several switches and behaviors are altered in the **sqlcmd** (Go) utility. For the most up-to-date list of missing flags for backward compatibility, visit the Prioritize implementation of back-compat flags GitHub discussion.

- In earlier versions of **sqlcmd** (Go), the `-P` switch was temporarily removed, and passwords for SQL Server Authentication could only be provided through these mechanisms:
  - The `SQLCMDPASSWORD` environment variable
  - The `:CONNECT` command
  - When prompted, the user could type the password to complete a connection
- `-r` requires a `0` or `1` argument
- `-R` switch is removed.
- `-I` switch is removed. To disable quoted identifier behavior, add `SET QUOTED IDENTIFIER OFF` in your scripts.
- `-N` now takes a string value that can be one of `true`, `false`, or `disable` to specify the encryption choice. (`default` is the same as omitting the parameter)
  - If `-N` and `-C` aren't provided, **sqlcmd** negotiates authentication with the server without validating the server certificate.
  - If `-N` is provided but `-C` isn't, **sqlcmd** requires validation of the server certificate. A `false` value for encryption could still lead to the encryption of the login packet.
  - If both `-N` and `-C` are provided, **sqlcmd** uses their values for encryption negotiation.
  - More information about client/server encryption negotiation can be found at MS-TDS PRELOGIN.

- `-u` The generated Unicode output file has the UTF-16 Little-Endian Byte-order mark (BOM) written to it.
- Some behaviors that were kept to maintain compatibility with `OSQL` may be changed, such as alignment of column headers for some data types.
- All commands must fit on one line, even `EXIT`. Interactive mode doesn't check for open parentheses or quotes for commands, and doesn't prompt for successive lines. This behavior is different to the ODBC version, which allows the query run by `EXIT(query)` to span multiple lines.

Connections from the **sqlcmd** (Go) utility are limited to TCP connections. Named pipes aren't supported at this time in the `go-mssqldb` driver.

Enhancements

- `:Connect` now has an optional `-G` parameter to select one of the authentication methods for Azure SQL Database - `SqlAuthentication`, `ActiveDirectoryDefault`, `ActiveDirectoryIntegrated`, `ActiveDirectoryServicePrincipal`, `ActiveDirectoryManagedIdentity`, `ActiveDirectoryPassword`. For more information, see Microsoft Entra authentication. If `-G` isn't provided, Integrated security or SQL Server authentication is used, depending on the presence of a `-U` user name parameter.
- The new `--driver-logging-level` command line parameter allows you to see traces from the `go-mssqldb` driver. Use `64` to see all traces.
- **sqlcmd** can now print results using a vertical format. Use the new `-F vertical` command line switch to set it. The

`SQLCMDFORMAT` scripting variable also controls it.

Command-line options

Login-related options

-A

Signs in to SQL Server with a dedicated administrator connection (DAC). This kind of connection is used to troubleshoot a server. This connection works only with server computers that support DAC. If DAC isn't available, **sqlcmd** generates an error message, and then exits. For more information about DAC, see Diagnostic Connection for Database Administrators. The `-A` option isn't supported with the `-G` option. When connecting to Azure SQL Database using `-A`, you must be an administrator on the logical SQL server. DAC isn't available for a Microsoft Entra administrator.

-C

This option is used by the client to configure it to implicitly trust the server certificate without validation. This option is equivalent to the ADO.NET option `TRUSTSERVERCERTIFICATE = true`.

For the **sqlcmd** (Go) utility, the following conditions also apply:

- If `-N` and `-C` aren't provided, **sqlcmd** negotiates authentication with the server without validating the server certificate.
- If `-N` is provided but `-C` isn't, **sqlcmd** requires validation of the server certificate. A `false` value for encryption could still lead to the encryption of the login packet.
- If both `-N` and `-C` are provided, **sqlcmd** uses their values for encryption negotiation.

-d *db_name*

Issues a `USE <db_name>` statement when you start **sqlcmd**. This option sets the **sqlcmd** scripting variable `SQLCMDDBNAME`. This parameter specifies the initial database. The default is your login's default-database property. If the database doesn't exist, an error message is generated and **sqlcmd** exits.

-D

Interprets the server name provided to `-S` as a DSN instead of a hostname. For more information, see *DSN support in sqlcmd and bcp* in Connecting with sqlcmd.

Note

The `-D` option is only available on Linux and macOS clients. On Windows clients, it previously referred to a now-obsolete option which has been removed and is ignored.

-l *login_timeout*

Specifies the number of seconds before a **sqlcmd** login to the ODBC driver times out when you try to connect to a server. This option sets the **sqlcmd** scripting variable `SQLCMDLOGINTIMEOUT`. The default time-out for login to **sqlcmd** is 8 seconds. When using the `-G` option to connect to Azure SQL Database or Azure Synapse Analytics and authenticate using Microsoft Entra ID, a timeout value of at least 30 seconds is recommended. The login time-out must be a number between `0` and `65534`. If the value supplied isn't numeric, or doesn't fall into that range, **sqlcmd** generates an error message. A value of `0` specifies time-out to be infinite.

-E

Uses a trusted connection instead of using a user name and password to sign in to SQL Server. By default, without `-E` specified, **sqlcmd** uses the trusted connection option.

The `-E` option ignores possible user name and password environment variable settings such as `SQLCMDPASSWORD`. If the `-E` option is used together with the `-U` option or the `-P` option, an error message is generated.

-g

Sets the Column Encryption setting to `Enabled`. For more information, see Always Encrypted. Only master keys stored in Windows Certificate Store are supported. The `-g` option requires at least **sqlcmd** version 13.1. To determine your version, execute `sqlcmd -?`.

-G

This option is used by the client when connecting to Azure SQL Database or Azure Synapse Analytics to specify that the user be authenticated using Microsoft Entra authentication. This option sets the **sqlcmd** scripting variable `SQLCMDUSEAAD = true`. The `-G` option requires at least **sqlcmd** version 13.1. To determine your version, execute `sqlcmd -?`. For more information, see Connecting to SQL Database or Azure Synapse Analytics By Using Microsoft Entra authentication. The `-A` option isn't supported with the `-G` option.

The `-G` option only applies to Azure SQL Database and Azure Synapse Analytics.

Microsoft Entra interactive authentication isn't currently supported on Linux or macOS. Microsoft Entra integrated authentication requires Microsoft ODBC Driver 17 for SQL Server version 17.6.1 or higher and a properly Configured Kerberos environment.

For more information about Microsoft Entra authentication, see Microsoft Entra authentication in sqlcmd.

-H *workstation_name*

A workstation name. This option sets the **sqlcmd** scripting variable `SQLCMDWORKSTATION`. The workstation name is listed in the `hostname` column of the `sys.sysprocesses` catalog view, and can be returned using the stored procedure `sp_who`. If this option isn't specified, the default is the current computer name. This name can be used to identify different **sqlcmd** sessions.

-j

Prints raw error messages to the screen.

-K *application_intent*

Declares the application workload type when connecting to a server. The only currently supported value is `ReadOnly`. If `-K` isn't specified, **sqlcmd** doesn't support connectivity to a secondary replica in an availability group. For more information, see Active Secondaries: Readable Secondary Replica (Always On Availability Groups).

-M *multisubnet_failover*

Always specify `-M` when connecting to the availability group listener of a SQL Server availability group or a SQL Server Failover Cluster Instance. `-M` provides for faster detection of and connection to the (currently) active server. If `-M` isn't specified, `-M` is off. For more information about Listeners, Client Connectivity, Application Failover, Creation and Configuration of Availability Groups (SQL Server), Failover Clustering and Always On Availability Groups (SQL Server), and Active

Secondaries: Readable Secondary Replicas(Always On Availability Groups).

-N

This option is used by the client to request an encrypted connection.

For the **sqlcmd** (Go) utility, `-N` now takes a string value that can be one of `true`, `false`, or `disable` to specify the encryption choice. (`default` is the same as omitting the parameter):

- If `-N` and `-C` aren't provided, **sqlcmd** negotiates authentication with the server without validating the server certificate.
- If `-N` is provided but `-C` isn't, **sqlcmd** requires validation of the server certificate. A `false` value for encryption could still lead to the encryption of the login packet.
- If both `-N` and `-C` are provided, **sqlcmd** uses their values for encryption negotiation.

-P *password*

A user-specified password. Passwords are case-sensitive. If the `-U` option is used and the `-P` option isn't used, and the `SQLCMDPASSWORD` environment variable hasn't been set, **sqlcmd** prompts the user for a password. We don't recommend the use of a null (blank) password, but you can specify the null password by using a pair of contiguous double-quotation marks for the parameter value (`""`).

Important

Using `-P` should be considered insecure. Avoid giving the password on the command line. Alternatively, use the `SQLCMDPASSWORD` environment variable, or interactively input the password by omitting the `-P` option.

We recommend that you use a strong password.

The password prompt is displayed by printing the password prompt to the console, as follows: `Password:`

User input is hidden. This means that nothing is displayed and the cursor stays in position.

The `SQLCMDPASSWORD` environment variable lets you set a default password for the current session. Therefore, passwords don't have to be hard-coded into batch files. The following example first sets the `SQLCMDPASSWORD` variable at the command prompt and then accesses the **sqlcmd** utility.

At the command prompt, type:

```
SET SQLCMDPASSWORD=p@a$$w0rd
sqlcmd
```

```
SET SQLCMDPASSWORD=p@a$$w0rd
sqlcmd
```

```
SET SQLCMDPASSWORD=p@a$$w0rd
sqlcmd
```

If the user name and password combination is incorrect, an error message is generated.

Note

The `OSQLPASSWORD` environment variable has been kept for backward compatibility. The `SQLCMDPASSWORD` environment variable takes precedence over the `OSQLPASSWORD` environment variable. This means that **sqlcmd** and **osql** can be used next to each other without interference. Old scripts will continue to work.

If the `-P` option is used with the `-E` option, an error message is generated.

If the `-P` option is followed by more than one argument, an error message is generated and the program exits.

A password containing special characters can generate an error message. You should escape special characters when using `-P`, or use the `SQLCMDPASSWORD` environment variable instead.

-S [*protocol*:]*server*[\instance_name][,*port*]

Specifies the instance of SQL Server to which to connect. It sets the **sqlcmd** scripting variable `SQLCMDSERVER`.

Specify *server_name* to connect to the default instance of SQL Server on that server computer. Specify *server_name*[\instance_name] to connect to a named instance of SQL Server on that server computer. If no server computer is specified, **sqlcmd** connects to the default instance of SQL Server on the local computer. This option is required when you execute **sqlcmd** from a remote computer on the network.

protocol can be `tcp` (TCP/IP), `lpc` (shared memory), or `np` (named pipes).

If you don't specify a *server_name*[\instance_name] when you start **sqlcmd**, SQL Server checks for and uses the `SQLCMDSERVER` environment variable.

> **Note**
>
> The `OSQLSERVER` environment variable has been kept for backward compatibility. The `SQLCMDSERVER` environment variable takes precedence over the `OSQLSERVER` environment variable. This means that **sqlcmd** and **osql** can be used next to each other without interference. Old scripts will continue to work.

-U *login_id*

The login name or contained database user name. For contained database users, you must provide the database name option (`-d`).

> **Note**
>
> The `OSQLUSER` environment variable has been kept for backward compatibility. The `SQLCMDUSER` environment variable takes precedence over the `OSQLUSER` environment variable. This means that **sqlcmd** and **osql** can be used next to each other without interference. Old scripts will continue to work.

If you don't specify either the `-U` option or the `-P` option, **sqlcmd** tries to connect by using Windows Authentication mode. Authentication is based on the Windows account of the user who is running **sqlcmd**.

If the `-U` option is used with the `-E` option (described later in this article), an error message is generated. If the `-U` option is followed by more than one argument, an error message is generated and the program exits.

-z *new_password*

Change the password:

```
sqlcmd -U someuser -P s0mep@ssword -z a_new_p@a$$w0rd
```

```
sqlcmd -U someuser -P s0mep@ssword -z a_new_p@a$$w0rd
```

```
sqlcmd -U someuser -P s0mep@ssword -z a_new_p@a$$w0rd
```

-Z *new_password*

Change the password and exit:

```
sqlcmd -U someuser -P s0mep@ssword -Z a_new_p@a$$w0rd
```

```
sqlcmd -U someuser -P s0mep@ssword -Z a_new_p@a$$w0rd
```

```
sqlcmd -U someuser -P s0mep@ssword -Z a_new_p@a$$w0rd
```

Input/output options

-f *codepage* | i:*codepage*[,o:*codepage*] | o:*codepage*[,i:*codepage*]

Specifies the input and output code pages. The codepage number is a numeric value that specifies an installed Windows code page.

Code-page conversion rules:

- If no code pages are specified, **sqlcmd** uses the current code page for both input and output files, unless the input file is a Unicode file, in which case no conversion is required.
- **sqlcmd** automatically recognizes both big-endian and little-endian Unicode input files. If the `-u` option has been specified, the output is always little-endian Unicode.
- If no output file is specified, the output code page is the console code page. This approach enables the output to be displayed correctly on the console.
- Multiple input files are assumed to be of the same code page. Unicode and non-Unicode input files can be mixed.

Enter `chcp` at the command prompt to verify the code page of `cmd.exe`.

-i *input_file*[,*input_file2*...]

Identifies the file that contains a batch of Transact-SQL statements or stored procedures. Multiple files may be specified that are read and processed in order. Don't use any spaces between file names. **sqlcmd** checks first to see whether all the specified files exist. If one or more files don't exist, **sqlcmd** exits. The `-i` and the `-Q`/`-q` options are mutually exclusive.

Path examples:

```
-i C:\<filename>
-i \\<Server>\<Share$>\<filename>
-i "C:\Some Folder\<file name>"
```

File paths that contain spaces must be enclosed in quotation marks.

This option may be used more than once:

```
sqlcmd -i <input_file1> -i <input_file2>
```

```
sqlcmd -i <input_file1> -i <input_file2>
```

```
sqlcmd -i <input_file1> -i <input_file2>
```

-o *output_file*

Identifies the file that receives output from **sqlcmd**.

If `-u` is specified, the *output_file* is stored in Unicode format. If the file name isn't valid, an error message is generated, and **sqlcmd** exits. **sqlcmd** doesn't support concurrent writing of multiple **sqlcmd** processes to the same file. The file output is corrupted or incorrect. The `-f` option is also relevant to file formats. This file is created if it doesn't exist. A file of the same name from a prior **sqlcmd** session is overwritten. The file specified here isn't the `stdout` file. If a `stdout` file is specified, this file isn't used.

Path examples:

```
-o C:< filename>
-o \\<Server>\<Share$>\<filename>
-o "C:\Some Folder\<file name>"
```

File paths that contain spaces must be enclosed in quotation marks.

-r[0 | 1]

Redirects the error message output to the screen (`stderr`). If you don't specify a parameter or if you specify `0`, only error messages that have a severity level of 11 or higher are redirected. If you specify `1`, all error message output including `PRINT` is redirected. This option has no effect if you use `-o`. By default, messages are sent to `stdout`.

Note

For the **sqlcmd** (Go) utility, `-r` requires a `0` or `1` argument.

-R

Applies to: ODBC **sqlcmd** only.

Causes **sqlcmd** to localize numeric, currency, date, and time columns retrieved from SQL Server based on the client's locale. By default, these columns are displayed using the server's regional settings.

-u

Specifies that *output_file* is stored in Unicode format, regardless of the format of *input_file*.

> **Note**
>
> For the **sqlcmd** (Go) utility, the generated Unicode output file has the UTF-16 Little-Endian Byte-order mark (BOM) written to it.

Query execution options

-e

Writes input scripts to the standard output device (`stdout`).

-I

Applies to: ODBC **sqlcmd** only.

Sets the `SET QUOTED_IDENTIFIER` connection option to `ON`. By default, it's set to `OFF`. For more information, see SET QUOTED_IDENTIFIER (Transact-SQL).

> **Note**
>
> To disable quoted identifier behavior in the **sqlcmd** (Go) utility, add `SET QUOTED IDENTIFIER OFF` in your scripts.

-q "*cmdline query*"

Executes a query when **sqlcmd** starts, but doesn't exit **sqlcmd** when the query has finished running. Multiple-semicolon-delimited queries can be executed. Use quotation marks around the query, as shown in the following example.

At the command prompt, type:

```
sqlcmd -d AdventureWorks2022 -q "SELECT FirstName, LastName

sqlcmd -d AdventureWorks2022 -q "SELECT TOP 5 FirstName FRON
```

```
sqlcmd -d AdventureWorks2022 -q "SELECT FirstName, LastName

sqlcmd -d AdventureWorks2022 -q "SELECT TOP 5 FirstName FRON
```

```
sqlcmd -d AdventureWorks2022 -q "SELECT FirstName, LastName

sqlcmd -d AdventureWorks2022 -q "SELECT TOP 5 FirstName FRON
```

Important

Don't use the `GO` terminator in the query.

If `-b` is specified together with this option, **sqlcmd** exits on error. `-b` is described elsewhere in this article.

-Q "*cmdline query*"

Executes a query when **sqlcmd** starts and then immediately exits **sqlcmd**. Multiple-semicolon-delimited queries can be executed.

Use quotation marks around the query, as shown in the following example.

At the command prompt, type:

```
sqlcmd -d AdventureWorks2022 -Q "SELECT FirstName, LastName

sqlcmd -d AdventureWorks2022 -Q "SELECT TOP 5 FirstName FROM
```

```
sqlcmd -d AdventureWorks2022 -Q "SELECT FirstName, LastName

sqlcmd -d AdventureWorks2022 -Q "SELECT TOP 5 FirstName FROM
```

```
sqlcmd -d AdventureWorks2022 -Q "SELECT FirstName, LastName

sqlcmd -d AdventureWorks2022 -Q "SELECT TOP 5 FirstName FROM
```

Important

Don't use the `GO` terminator in the query.

If `-b` is specified together with this option, **sqlcmd** exits on error. `-b` is described elsewhere in this article.

-t *query_timeout*

Specifies the number of seconds before a command (or Transact-SQL statement) times out. This option sets the **sqlcmd** scripting variable `SQLCMDSTATTIMEOUT`. If a *query_timeout* value isn't specified, the command doesn't time out. The *query_timeout* must be a number between `1` and `65534`. If the value supplied isn't numeric or doesn't fall into that range, **sqlcmd** generates an error message.

Note

The actual time out value may vary from the specified *query_timeout* value by several seconds.

-v var = *value* [var = *value*...]

Creates a **sqlcmd** scripting variable that can be used in a **sqlcmd** script. Enclose the value in quotation marks if the value contains spaces. You can specify multiple `<var>="<value>"` values. If there are errors in any of the values specified, **sqlcmd** generates an error message and then exits.

```
sqlcmd -v MyVar1=something MyVar2="some thing"

sqlcmd -v MyVar1=something -v MyVar2="some thing"
```

```
sqlcmd -v MyVar1=something MyVar2="some thing"

sqlcmd -v MyVar1=something -v MyVar2="some thing"
```

```
sqlcmd -v MyVar1=something MyVar2="some thing"

sqlcmd -v MyVar1=something -v MyVar2="some thing"
```

-x

Causes **sqlcmd** to ignore scripting variables. This parameter is useful when a script contains many `INSERT` statements that may contain strings that have the same format as regular variables, such as `$(<variable_name>)`.

Format options

-h *headers*

Specifies the number of rows to print between the column headings. The default is to print headings one time for each set of query results. This option sets the **sqlcmd** scripting variable `SQLCMDHEADERS`. Use `-1` to specify that headers not be printed. Any value that isn't valid causes **sqlcmd** to generate an error message and then exit.

-k [1 | 2]

Removes all control characters, such as tabs and new line characters from the output. This parameter preserves column formatting when data is returned.

- `-k` removes control characters.
- `-k1` replaces each control character with a space.
- `-k2` replaces consecutive control characters with a single space.

-s *col_separator*

Specifies the column-separator character. The default is a blank space. This option sets the **sqlcmd** scripting variable `SQLCMDCOLSEP`. To use characters that have special meaning to the operating system, such as the ampersand (`&`) or semicolon (`;`), enclose the character in quotation marks (`"`). The column separator can be any 8-bit character.

-w *screen_width*

Specifies the screen width for output. This option sets the **sqlcmd** scripting variable `SQLCMDCOLWIDTH`. The column width must be a number greater than `8` and less than `65536`. If the specified column width doesn't fall into that range, **sqlcmd** generates an error message. The default width is 80 characters. When an output line exceeds the specified column width, it wraps on to the next line.

-W

This option removes trailing spaces from a column. Use this option together with the `-s` option when preparing data that is to be exported to another application. Can't be used with the `-y` or `-Y` options.

-y *variable_length_type_display_width*

Sets the **sqlcmd** scripting variable `SQLCMDMAXVARTYPEWIDTH`. The default is `256`. It limits the number of characters that are returned for the large variable length data types:

- **varchar(max)**
- **nvarchar(max)**
- **varbinary(max)**
- **xml**
- **user-defined data types** (UDTs)
- **text**
- **ntext**
- **image**

UDTs can be of fixed length depending on the implementation. If this length of a fixed length UDT is shorter that *display_width*, the value of the UDT returned isn't affected. However, if the length is longer than *display_width*, the output is truncated.

ⓧ **Caution**

Use the `-y 0` option with extreme caution, because it may cause significant performance issues on both the server and the network, depending on the size of data returned.

-Y *fixed_length_type_display_width*

Sets the **sqlcmd** scripting variable `SQLCMDMAXFIXEDTYPEWIDTH`. The default is `0` (unlimited). Limits the number of characters that are returned for the following data types:

- **char(*n*)**, where 1 <= *n* <= 8000
- **nchar(*n*)**, where 1 <= *n* <= 4000
- **varchar(*n*)**, where 1 <= *n* <= 8000
- **nvarchar(*n*)**, where 1 <= *n* <= 4000
- **varbinary(*n*)**, where 1 <= *n* <= 4000
- **sql_variant**

Error reporting options

-b

Specifies that **sqlcmd** exits and returns a `DOS ERRORLEVEL` value when an error occurs. The value that is returned to the `ERRORLEVEL` variable is `1` when the SQL Server error message has a severity level greater than 10; otherwise, the value returned is `0`. If the `-V` option has been set in addition to `-b`, **sqlcmd** doesn't report an error if the severity level is lower than the values set using `-V`. Command prompt batch files can test the value of `ERRORLEVEL` and handle the error appropriately. **sqlcmd** doesn't report errors for severity level 10 (informational messages).

If the **sqlcmd** script contains an incorrect comment, syntax error, or is missing a scripting variable, the `ERRORLEVEL` returned is `1`.

-m *error_level*

Controls which error messages are sent to `stdout`. Messages that have a severity level greater than or equal to this level are sent. When this value is set to `-1`, all messages including informational messages, are sent. Spaces aren't allowed between the `-m` and `-1`. For example, `-m-1` is valid, and `-m -1` isn't.

This option also sets the **sqlcmd** scripting variable `SQLCMDERRORLEVEL`. This variable has a default of `0`.

-V *error_severity_level*

Controls the severity level that is used to set the `ERRORLEVEL` variable. Error messages that have severity levels greater than or equal to this value set `ERRORLEVEL`. Values that are less than 0 are reported as `0`. Batch and CMD files can be used to test the value of the `ERRORLEVEL` variable.

Miscellaneous options

-a *packet_size*

Requests a packet of a different size. This option sets the **sqlcmd** scripting variable `SQLCMDPACKETSIZE`. *packet_size* must be a value between `512` and `32767`. The default is `4096`. A larger packet size can enhance performance for execution of scripts that have lots of Transact-SQL statements between `GO` commands. You can request a larger packet size. However, if the request is denied, **sqlcmd** uses the server default for packet size.

-c *batch_terminator*

Specifies the batch terminator. By default, commands are terminated and sent to SQL Server by typing the word `GO` on a line by itself. When you reset the batch terminator, don't use Transact-SQL reserved keywords or characters that have special meaning to the operating system, even if they're preceded by a backslash.

-L[c]

Lists the locally configured server computers, and the names of the server computers that are broadcasting on the network. This parameter can't be used in combination with other parameters. The maximum number of server computers that can be listed is 3000. If the server list is truncated because of the size of the buffer a warning message is displayed.

> **Note**
>
> Because of the nature of broadcasting on networks, **sqlcmd** may not receive a timely response from all servers. Therefore, the list of servers returned may vary for each invocation of this option.

If the optional parameter `c` is specified, the output appears without the `Servers:` header line, and each server line is listed without leading spaces. This presentation is referred to as clean output. Clean output improves the processing performance of scripting languages.

-p[1]

Prints performance statistics for every result set. The following display is an example of the format for performance statistics:

```
Network packet size (bytes): n

x xact[s]:

Clock Time (ms.): total       t1  avg       t2 (t3 xacts per
```

Where:

- `x` = Number of transactions that are processed by SQL Server.
- `t1` = Total time for all transactions.

- `t2` = Average time for a single transaction.
- `t3` = Average number of transactions per second.

All times are in milliseconds.

If the optional parameter `1` is specified, the output format of the statistics is in colon-separated format that can be imported easily into a spreadsheet or processed by a script.

If the optional parameter is any value other than `1`, an error is generated and **sqlcmd** exits.

-X[1]

Disables commands that might compromise system security when **sqlcmd** is executed from a batch file. The disabled commands are still recognized; **sqlcmd** issues a warning message and continues. If the optional parameter `1` is specified, **sqlcmd** generates an error message and then exits. The following commands are disabled when the `-X` option is used:

- `ED`
- `!!` *command*

If the `-X` option is specified, it prevents environment variables from being passed on to **sqlcmd**. It also prevents the startup script specified by using the `SQLCMDINI` scripting variable from being executed. For more information about **sqlcmd** scripting variables, see sqlcmd - Use with scripting variables.

-?

Displays the version of **sqlcmd** and a syntax summary of **sqlcmd** options.

ⓘ **Note**

On macOS, run `sqlcmd '-?'` (with quotation marks) instead.

Remarks

Options don't have to be used in the order shown in the syntax section.

When multiple results are returned, **sqlcmd** prints a blank line between each result set in a batch. In addition, the `<x> rows affected` message doesn't appear when it doesn't apply to the statement executed.

To use **sqlcmd** interactively, type `sqlcmd` at the command prompt with any one or more of the options described earlier in this article. For more information, see Use the sqlcmd Utility

Note

The options `-l`, `-Q`, `-Z` or `-i` cause **sqlcmd** to exit after execution.

The total length of the **sqlcmd** command-line in the command environment (for example `cmd.exe` or `bash`), including all arguments and expanded variables, is determined by the underlying operating system.

Variable precedence (low to high)

1. System-level environmental variables
2. User-level environmental variables
3. Command shell (`SET X=Y`) set at command prompt before running **sqlcmd**
4. `sqlcmd -v X=Y`
5. `:Setvar X Y`

Note

To view the environmental variables, in **Control Panel**, open **System**, and then select the **Advanced** tab.

sqlcmd scripting variables

Expand table

Variable	Related option	R/W	Default
SQLCMDUSER	-U	R	""
SQLCMDPASSWORD	-P	--	""
SQLCMDSERVER	-S	R	"DefaultLocalInstance"
SQLCMDWORKSTATION	-H	R	"ComputerName"
SQLCMDDBNAME	-d	R	""
SQLCMDLOGINTIMEOUT	-l	R/W	"8" (seconds)
SQLCMDSTATTIMEOUT	-t	R/W	"0" = wait indefinitely
SQLCMDHEADERS	-h	R/W	"0"
SQLCMDCOLSEP	-s	R/W	" "
SQLCMDCOLWIDTH	-w	R/W	"0"
SQLCMDPACKETSIZE	-a	R	"4096"
SQLCMDERRORLEVEL	-m	R/W	0
SQLCMDMAXVARTYPEWIDTH	-y	R/W	"256"
SQLCMDMAXFIXEDTYPEWIDTH	-Y	R/W	"0" = unlimited
SQLCMDEDITOR		R/W	"edit.com"

SQLCMDINI		R	""
SQLCMDUSEAAD	-G	R/W	""

`SQLCMDUSER`, `SQLCMDPASSWORD`, and `SQLCMDSERVER` are set when `:Connect` is used.

`R` indicates the value can only be set one time during program initialization.

`R/W` indicates that the value can be modified by using the `:setvar` command and subsequent commands are influenced by the new value.

sqlcmd commands

In addition to Transact-SQL statements within **sqlcmd**, the following commands are also available:

`GO` [*count*]

`:List`

`[:]RESET`

`:Error`

`[:]ED`

`:Out`

`[:]!!`

`:Perftrace`

`[:]QUIT`

`:Connect`

`[:]EXIT`

`:On Error`

`:r`

`:Help`

`:ServerList`

`:XML [ ON | OFF ]`

`:Setvar`

`:Listvar`

Be aware of the following when you use **sqlcmd** commands:

- All **sqlcmd** commands, except `GO`, must be prefixed by a colon (`:`).

  > **Important**
  >
  > To maintain backward compatibility with existing **osql** scripts, some of the commands will be recognized without the colon, indicated by the `:`.

- **sqlcmd** commands are recognized only if they appear at the start of a line.
- All **sqlcmd** commands are case insensitive.
- Each command must be on a separate line. A command can't be followed by a Transact-SQL statement or another command.

- Commands are executed immediately. They aren't put in the execution buffer as Transact-SQL statements are.

Editing commands

[:]ED

Starts the text editor. This editor can be used to edit the current Transact-SQL batch, or the last executed batch. To edit the last executed batch, the `ED` command must be typed immediately after the last batch has completed execution.

The text editor is defined by the `SQLCMDEDITOR` environment variable. The default editor is `Edit`. To change the editor, set the `SQLCMDEDITOR` environment variable. For example, to set the editor to Microsoft Notepad, at the command prompt, type:

`SET SQLCMDEDITOR=notepad`

[:]RESET

Clears the statement cache.

:List

Prints the content of the statement cache.

Variables

:Setvar <var> ["*value*"]

Defines **sqlcmd** scripting variables. Scripting variables have the following format: `$(VARNAME)`.

Variable names are case insensitive.

Scripting variables can be set in the following ways:

- Implicitly using a command-line option. For example, the `-l` option sets the `SQLCMDLOGINTIMEOUT` **sqlcmd** variable.
- Explicitly by using the `:Setvar` command.
- By defining an environment variable before you run **sqlcmd**.

> **Note**
>
> The `-X` option prevents environment variables from being passed on to **sqlcmd**.

If a variable defined by using `:Setvar` and an environment variable have the same name, the variable defined by using `:Setvar` takes precedence.

Variable names must not contain blank space characters.

Variable names can't have the same form as a variable expression, such as `$(var)`.

If the string value of the scripting variable contains blank spaces, enclose the value in quotation marks. If a value for a scripting variable isn't specified, the scripting variable is dropped.

:Listvar

Displays a list of the scripting variables that are currently set.

> **Note**
>
> Only scripting variables that are set by **sqlcmd**, and those that are set using the `:Setvar` command will be displayed.

Output commands

:Error <*filename*> | STDERR | STDOUT

Redirect all error output to the file specified by *filename*, to `stderr` or to `stdout`. The `:Error` command can appear multiple times in a script. By default, error output is sent to `stderr`.

- *filename*

  Creates and opens a file that receives the output. If the file already exists, it's truncated to zero bytes. If the file isn't available because of permissions or other reasons, the output isn't switched, and is sent to the last specified or default destination.

- STDERR

  Switches error output to the `stderr` stream. If this has been redirected, the target to which the stream has been redirected receives the error output.

- STDOUT

  Switches error output to the `stdout` stream. If this has been redirected, the target to which the stream has been redirected receives the error output.

:Out <*filename*> | STDERR | STDOUT

Creates and redirects all query results to the file specified by *file name*, to `stderr` or to `stdout`. By default, output is sent to `stdout`. If the file already exists, it's truncated to zero bytes. The `:Out` command can appear multiple times in a script.

:Perftrace <*filename*> | STDERR | STDOUT

Creates and redirects all performance trace information to the file specified by *file name*, to `stderr` or to `stdout`. By default performance trace output is sent to `stdout`. If the file already exists, it's truncated to zero bytes. The `:Perftrace` command can appear multiple times in a script.

Execution control commands

:On Error [exit | ignore]

Sets the action to be performed when an error occurs during script or batch execution.

When the `exit` option is used, **sqlcmd** exits with the appropriate error value.

When the `ignore` option is used, **sqlcmd** ignores the error and continues executing the batch or script. By default, an error message is printed.

[:]QUIT

Causes **sqlcmd** to exit.

[:]EXIT [(*statement*)]

Lets you use the result of a `SELECT` statement as the return value from **sqlcmd**. If numeric, the first column of the last result row is converted to a 4-byte integer (long). MS-DOS, Linux, and macOS pass the low byte to the parent process or operating system error level. Windows 2000 and later versions passes the whole 4-byte integer. The syntax is `:EXIT(query)`.

For example:

```
:EXIT(SELECT @@ROWCOUNT)
```

You can also include the `:EXIT` parameter as part of a batch file. For example, at the command prompt, type:

`sqlcmd -Q ":EXIT(SELECT COUNT(*) FROM '%1')"`

The **sqlcmd** utility sends everything between the parentheses (`()`) to the server. If a system stored procedure selects a set and returns a value, only the selection is returned. The `:EXIT()` statement with nothing between the parentheses executes everything before it in the batch, and then exits without a return value.

When an incorrect query is specified, **sqlcmd** exits without a return value.

Here is a list of `EXIT` formats:

- `:EXIT`

  Doesn't execute the batch, and then quits immediately and returns no value.

- `:EXIT( )`

  Executes the batch, and then quits and returns no value.

- `:EXIT(query)`

  Executes the batch that includes the query, and then quits after it returns the results of the query.

If `RAISERROR` is used within a **sqlcmd** script, and a state of 127 is raised, **sqlcmd** quits, and returns the message ID back to the client. For example:

```
RAISERROR(50001, 10, 127)
```

This error causes the **sqlcmd** script to end and return the message ID 50001 to the client.

The return values `-1` to `-99` are reserved by SQL Server, and **sqlcmd** defines the following additional return values:

Expand table

Return value	Description
-100	Error encountered prior to selecting return value.
-101	No rows found when selecting return value.
-102	Conversion error occurred when selecting return value.

GO [*count*]

`GO` signals both the end of a batch and the execution of any cached Transact-SQL statements. The batch is executed multiple times as separate batches. You can't declare a variable more than once in a single batch.

Miscellaneous commands

:r <*filename*>

Parses additional Transact-SQL statements and **sqlcmd** commands from the file specified by *filename* into the statement cache. *filename* is read relative to the startup directory in which **sqlcmd** was run.

If the file contains Transact-SQL statements that aren't followed by `GO`, you must enter `GO` on the line that follows `:r`.

The file will be read and executed after a batch terminator is encountered. You can issue multiple `:r` commands. The file may include any **sqlcmd** command, including the batch terminator `GO`.

> **Note**
>
> The line count that is displayed in interactive mode will be increased by one for every `:r` command encountered. The `:r` command will appear in the output of the list command.

:ServerList

Lists the locally configured servers and the names of the servers broadcasting on the network.

:Connect *server_name*[\instance_name] [-l *timeout*] [-U *user_name* [-P *password*]]

Connects to an instance of SQL Server. Also closes the current connection.

Time-out options:

Expand table

Value	Behavior
`0`	Wait forever
`n>0`	Wait for *n* seconds

The `SQLCMDSERVER` scripting variable reflects the current active connection.

If *timeout* isn't specified, the value of the `SQLCMDLOGINTIMEOUT` variable is the default.

If only *user_name* is specified (either as an option, or as an environment variable), the user is prompted to enter a password. Users aren't prompted if the `SQLCMDUSER` or `SQLCMDPASSWORD` environment variables have been set. If you don't provide options or environment variables, Windows Authentication mode is used to sign in. For example, to connect to an instance, `instance1`, of SQL Server, `myserver`, by using integrated security you would use the following command:

```
:connect myserver\instance1
```

To connect to the default instance of `myserver` using scripting variables, you would use the following settings:

```
:setvar myusername test
:setvar myservername myserver
:connect $(myservername) $(myusername)
```

[:]!! *command*

Executes operating system commands. To execute an operating system command, start a line with two exclamation marks (`!!`) followed by the operating system command. For example:

```
:!! dir
```

ⓘ **Note**

The command is executed on the computer on which **sqlcmd** is running.

:XML [ON | OFF]

For more information, see XML Output Format and JSON Output Format in this article.

:Help

Lists **sqlcmd** commands, together with a short description of each command.

sqlcmd file names

sqlcmd input files can be specified with the `-i` option or the `:r` command. Output files can be specified with the `-o` option or the `:Error`, `:Out` and `:Perftrace` commands. The following are some guidelines for working with these files:

- `:Error`, `:Out` and `:Perftrace` should use separate *filename* values. If the same *filename* is used, inputs from the commands may be intermixed.
- If an input file that is located on a remote server is called from **sqlcmd** on a local computer, and the file contains a drive file path such as `:Out c:\OutputFile.txt`, the output file is created on the local computer and not on the remote server.
- Valid file paths include: `C:\<filename>`, `\\<Server>\<Share$>\<filename>`, and `"C:\Some Folder\<file name>"`. If there's a space in the path, use quotation marks.
- Each new **sqlcmd** session overwrites existing files that have the same names.

Informational messages

sqlcmd prints any informational message that is sent by the server. In the following example, after the Transact-SQL statements are executed, an informational message is printed.

Start **sqlcmd**. At the **sqlcmd** command prompt, type the query:

```
USE AdventureWorks2022;
GO
```

When you press `ENTER`, the following informational message is printed:

```
Changed database context to 'AdventureWorks2022'.
```

Output format from Transact-SQL queries

sqlcmd first prints a column header that contains the column names specified in the select list. The column names are separated by using the `SQLCMDCOLSEP` character. By default, this is a space. If the column name is shorter than the column width, the output is padded with spaces up to the next column.

This line is followed by a separator line that is a series of dash characters. The following output shows an example.

Start **sqlcmd**. At the **sqlcmd** command prompt, type the query:

```
USE AdventureWorks2022;
SELECT TOP (2) BusinessEntityID, FirstName, LastName
FROM Person.Person;
GO
```

When you press `ENTER`, the following result set is returned.

```
BusinessEntityID FirstName    LastName
---------------- ------------ ----------
285              Syed         Abbas
293              Catherine    Abel
(2 row(s) affected)
```

Although the `BusinessEntityID` column is only four characters wide, it has been expanded to accommodate the longer column name. By default, output is terminated at 80 characters. This width can be changed by using the `-w` option, or by setting the `SQLCMDCOLWIDTH` scripting variable.

XML output format

XML output that is the result of a `FOR XML` clause is output, unformatted, in a continuous stream.

When you expect XML output, use the following command: `:XML ON`.

Note

sqlcmd returns error messages in the usual format. The error messages are also output in the XML text stream in XML format. By using `:XML ON`, **sqlcmd** does not display informational messages.

To set the XML mode to off, use the following command: `:XML OFF`.

The `GO` command shouldn't appear before the `:XML OFF` command is issued, because the `:XML OFF` command switches **sqlcmd** back to row-oriented output.

XML (streamed) data and rowset data can't be mixed. If the `:XML ON` command hasn't been issued before a Transact-SQL statement that outputs XML streams is executed, the output is garbled. Once the `:XML ON` command has been issued, you can't execute Transact-SQL statements that output regular row sets.

Note

The `:XML` command does not support the `SET STATISTICS XML` statement.

JSON output format

When you expect JSON output, use the following command: `:XML ON`. Otherwise, the output includes both the column name and the JSON text. This output isn't valid JSON.

To set the XML mode to off, use the following command: `:XML OFF`.

For more info, see XML Output Format in this article.

Use Microsoft Entra authentication

Examples using Microsoft Entra authentication:

```
sqlcmd -S Target_DB_or_DW.testsrv.database.windows.net  -G

sqlcmd -S Target_DB_or_DW.testsrv.database.windows.net -G -l
```

```
sqlcmd -S Target_DB_or_DW.testsrv.database.windows.net  -G

sqlcmd -S Target_DB_or_DW.testsrv.database.windows.net -G -l
```

```
sqlcmd -S Target_DB_or_DW.testsrv.database.windows.net  -G

sqlcmd -S Target_DB_or_DW.testsrv.database.windows.net -G -l
```

sqlcmd best practices

Use the following practices to help maximize security and efficiency.

- Use integrated security.
- Use `-X[1]` in automated environments.
- Secure input and output files by using appropriate file system permissions.
- To increase performance, do as much in one **sqlcmd** session as you can, instead of in a series of sessions.
- Set time-out values for batch or query execution higher than you expect it will take to execute the batch or query.

Use the following practices to help maximize correctness:

- Use `-V16` to log any severity 16 level messages. Severity 16 messages indicate general errors that can be corrected by the user.
- Check the exit code and `DOS ERRORLEVEL` variable after the process has exited. **sqlcmd** returns `0` normally, otherwise it sets the `ERRORLEVEL` as configured by `-V`. In other words, `ERRORLEVEL` shouldn't be expected to be the same value as the error number reported from SQL Server. The error number is a SQL Server-specific value corresponding to the system function **@@ERROR**. `ERRORLEVEL` is a **sqlcmd**-specific value to indicate why **sqlcmd** terminated, and its value is influenced by specifying `-b` command line argument.

Using `-V16` in combination with checking the exit code and `DOS ERRORLEVEL` can help catch errors in automated environments, particularly quality gates before a production release.

Related content

- Learn more about the new go-sqlcmd utility on GitHub
- Quickstart: Run SQL Server Linux container images with Docker
- sqlcmd - Start the utility
- sqlcmd - Run Transact-SQL script files
- sqlcmd - use the utility
- sqlcmd - Use with scripting variables
- sqlcmd - Connect to the database engine
- Edit SQLCMD Scripts with Query Editor
- Manage Job Steps
- Create a CmdExec Job Step

Feedback

Was this page helpful? Yes No

Provide product feedback | Get help at Microsoft Q&A