sqlcmd Utility

2017-4-10 • 32 min to read • Contributors  all

In this article

THIS TOPIC APPLIES TO: SQL Server (starting with 2008) Azure SQL Database Azure SQL Data Warehouse Parallel Data Warehouse

> For content related to previous versions of SQL Server, see sqlcmd Utility.

The **sqlcmd** utility lets you enter Transact-SQL statements, system procedures, and script files at the command prompt, in **Query Editor** in SQLCMD mode, in a Windows script file or in an operating system (Cmd.exe) job step of a SQL Server Agent job. This utility uses ODBC to execute Transact-SQL batches.

> **Note**
>
> The most recent versions of the sqlcmd utility is available as a web release from the Download Center. At least version 13.1 is required to support Always Encrypted (`-g`) and Azure Active Directory authentication (`-G`). (You may have several versions of sqlcmd.exe installed on your computer. Be sure you are using the correct version. To determine the version, execute `sqlcmd -?` .)

To run sqlcmd statements in SSMS, select SQLCMD Mode from the top navigation Query Menu dropdown.

> **Important**
>
> SQL Server Management Studio (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 SQL Server Management Studio in SQLCMD Mode and in the **sqlcmd** utility.

Currently, **sqlcmd** does not require a space between the command line option and the value. However, in a future release, a space may be required between the command line option and the value.

Other topics:

- Start the sqlcmd Utility
- Use the sqlcmd Utility

Syntax

Copy

```
sqlcmd
   -a packet_size
   -A (dedicated administrator connection)
   -b (terminate batch job if there is an error)
   -c batch_terminator
   -C (trust the server certificate)
   -d db_name
   -e (echo input)
   -E (use trusted connection)
   -f codepage | i:codepage[,o:codepage] | o:codepage[,i:codepage]
   -g (enable column encryption)
   -G (use Azure Active Directory for authentication)
   -h rows_per_header
   -H workstation_name
   -i input_file
   -I (enable quoted identifiers)
   -j (Print raw error messages)
   -k[1 | 2] (remove or replace control characters)
   -K application_intent
   -l login_timeout
   -L[c] (list servers, optional clean output)
   -m error_level
   -M multisubnet_failover
   -N (encrypt connection)
   -o output_file
   -p[1] (print statistics, optional colon format)
   -P password
   -q "cmdline query"
   -Q "cmdline query" (and exit)
   -r[0 | 1] (msgs to stderr)
   -R (use client regional settings)
   -s col_separator
   -S [protocol:]server[instance_name][,port]
   -t query_timeout
   -u (unicode output file)
   -U login_id
   -v var = "value"
   -V error_severity_level
   -w column_width
   -W (remove trailing spaces)
   -x (disable variable substitution)
   -X[1] (disable commands, startup script, environment variables, optional exit
   -y variable_length_type_display_width
```

```
    -Y fixed_length_type_display_width
    -z new_password
    -Z new_password (and exit)
    -? (usage)
```

Command-line Options

Login-Related Options

-A

Logs in to SQL Server with a Dedicated Administrator Connection (DAC). This kind of connection is used to troubleshoot a server. This will only work with server computers that support DAC. If DAC is not available, **sqlcmd** generates an error message and then exits. For more information about DAC, see Diagnostic Connection for Database Administrators. The -A option is not supported with the -G option. When connecting to SQL Database using -A, you must be a SQL server administrator. The DAC is not availble for an Azure Active Directory adminstrator.

-C

This switch 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`.

-d *db_name*

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

-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 eight seconds. When using the **-G** option to connect to SQL Database or SQL Data Warehouse and authenticate using Azure Active Directory, 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 is not numeric or does not 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 log on 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 switch requires at least **sqlcmd** version 13.1. To determine your version, execute `sqlcmd -?`.

-G

This switch is used by the client when connecting to SQL Database or SQL Data Warehouse to specify that the user be authenticated using Azure Active Directory authentication. This option sets the **sqlcmd** scripting variable SQLCMDUSEAAD = true. The -G switch requires at least **sqlcmd** version 13.1. To determine your version, execute `sqlcmd -?`. For more information, see Connecting to SQL Database or SQL Data Warehouse By Using Azure Active Directory Authentication. The -A option is not supported with the -G option.

> **Important**
>
> The **-G** option only applies to Azure SQL Database and Azure Data Warehouse.

- **Azure Active Directory Username and Password:**

  When you want to use an Azure Active Directory user name and password, you can provide the **-G** option and also use the user name and password by providing the **-U** and **-P** options.

  ```
  Sqlcmd -S Target_DB_or_DW.testsrv.database.windows.net -U bob@contoso.co
  ```

  This will generate the following connection string in the backend:

  ```
  SERVER = Target_DB_or_DW.testsrv.database.windows.net;UID= bob@contoso.
  ```

- **Azure Active Directory Integrated**

  For Azure Active Directory Integrated authentication, provide the **-G** option without a user name or password:

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

This will generate the following connection string in the backend:

Copy

```
SERVER = Target_DB_or_DW.testsrv.database.windows.net Authentication = A
```

> **Note**
>
> The -E option (Trusted_Connection) cannot be used along with the -G option).

-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 is not 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** is not specified, the sqlcmd utility will not support connectivity to a secondary replica in an Always On 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** is not specified, **-M** is off. For more information about Always On availability groups, see Availability Group Listeners, Client Connectivity, and Application Failover (SQL Server), Creation and Configuration of Availability Groups (SQL Server), [Failover Clustering and Always On Availability Groups (SQL Server)](https://msdn.microsoft.comlibrary/ff929171.aspx, and [Active Secondaries: Readable Secondary Replicas(Always On Availability Groups)](https://msdn.microsoft.com/library/ff878253.aspx.

-N
This switch is used by the client to request an encrypted connection.

-P *password*
Is a user-specified password. Passwords are case sensitive. If the -U option is used and the **-P**

option is not used, and the SQLCMDPASSWORD environment variable has not been set, **sqlcmd** prompts the user for a password. To specify a null password (not recommended) use **-P ""**. And remember to always:

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 do not 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
```

At the following command prompt, type:

```
sqlcmd
```

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

NOTE! The OSQLPASSWORD environment variable was 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 and that 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.

-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 do not 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 and that old scripts will continue to work.

-U *login_id*
Is 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 is available 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. It also means that existing **osql** scripts will continue to work.

If neither the **-U** option nor the **-P** option is specified, **sqlcmd** tries to connect by using Microsoft 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 topic), 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 password:

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

-Z *new_password*
Change password and exit:

```
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** will use 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 will always be little-endian Unicode.

- If no output file is specified, the output code page will be the console code page. This 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 SQL statements or stored procedures. Multiple files may be specified that will be read and processed in order. Do not use any spaces between file names. **sqlcmd** will first check to see whether all the specified files exist. If one or more files do not exist, **sqlcmd** will exit. 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: **-i***input_file* **-I***/ input_file.*

-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 is not valid, an error message is generated, and **sqlcmd** exits. **sqlcmd** does not support concurrent writing of multiple **sqlcmd** processes to the same file. The file output will be corrupted or incorrect. See the **-f** switch for more information about file formats. This file will be created if it does not exist. A file of the same name from a prior **sqlcmd** session will be overwritten. The file specified here is not the **stdout** file. If a **stdout** file is specified this file will not be used.

Path examples:

Copy

```
-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 do not 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. Has no effect if you use -o. By default, messages are sent to **stdout**.

-R
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*.

Query Execution Options
-e
Writes input scripts to the standard output device (**stdout**).

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

-q" *cmdline query* **"**
Executes a query when **sqlcmd** starts, but does not 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 AdventureWorks2012 -q "SELECT FirstName, LastName FROM Person.Person
WHERE LastName LIKE 'Whi%';"
```

```
sqlcmd -d AdventureWorks2012 -q "SELECT TOP 5 FirstName FROM Person.Person;SELECT
TOP 5 LastName FROM Person.Person;"
```

> **Important**
>
> Do not use the GO terminator in the query.

If **-b** is specified together with this option, **sqlcmd** exits on error. **-b** is described later in this topic.

-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 AdventureWorks2012 -Q "SELECT FirstName, LastName FROM Person.Person
WHERE LastName LIKE 'Whi%';"
```

```
sqlcmd -d AdventureWorks2012 -Q "SELECT TOP 5 FirstName FROM Person.Person;SELECT
TOP 5 LastName FROM Person.Person;"
```

> **Important**
>
> Do not use the GO terminator in the query.

If **-b** is specified together with this option, **sqlcmd** exits on error. **-b** is described later in this topic.

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

> **Note**
>
> The actual time out value may vary from the specified *time_out* value by several seconds.

-vvar = *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***=**"***values***"** 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"
```

-x
Causes **sqlcmd** to ignore scripting variables. This 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*).

Formatting 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 must not be printed. Any value that is not 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 preserves column formatting when data is returned. If 1 is specified, the control characters are replaced by a single space. If 2 is specified, consecutive control characters are replaced by a single space. **-k** is the same as **-k1**.

-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 *column_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 does not fall into that range, **sqlcmd** generates and 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. Cannot 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**
- **UDT (user-defined data types)**
- **text**
- **ntext**
- **image**

> **Note**
>
> 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 is not affected. However, if the length is longer than *display_width*, the output is truncated.

> **Important**
>
> Use the **-y 0** option with extreme caution because it may cause serious 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** *n* **)**, where 1<=n<=4000
- **varchar(n** *n* **)**, where 1<=n<=8000
- **nvarchar(n** *n* **)**, where 1<=n<=4000
- **varbinary(n** *n* **)**, where 1<=n<=4000
- **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 DOS 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** will not 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** does not report errors for severity level 10 (informational messages).

If the **sqlcmd** script contains an incorrect comment, syntax error, or is missing a scripting variable, 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 are not allowed between the **-m** and **-1**. For example, **-m-1** is valid, and **-m-1** is not.

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 = 4096. A larger packet size can enhance performance for execution of scripts that have lots of 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, do not use Transact-SQL reserved keywords or characters that have special meaning to the operating system, even if they are 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 cannot 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 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 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 sec.)
```

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 Use sqlcmd with Scripting Variables.

-?

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

Remarks

Options do not 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 does not appear when it does not 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 topic. 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 (Cmd.exe), including all arguments and expanded variables, is that which is determined by the operating system for Cmd.exe.

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 click the **Advanced** tab.

sqlcmd Scripting Variables

Variable	Related switch	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 will be 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. This is 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 cannot be followed by a Transact-SQL statement or another command.
- Commands are executed immediately. They are not 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 **-I** 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 cannot 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 is not 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 *file name*, to **stderr** or to **stdout**. The **Error** command can appear multiple times in a script. By default, error output is sent to **stderr**.

file name

Creates and opens a file that will receive the output. If the file already exists, it will be truncated to zero bytes. If the file is not available because of permissions or other reasons, the output will not be switched and will be 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 will receive 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 will receive 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 will be 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 will be 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 will be 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 passes

the low byte to the parent process or operating system error level. Windows 200x 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** will exit without a return value.

Here is a list of EXIT formats:

- :EXIT

  Does not 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** will quit and return the message ID back to the client. For example:

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

  This error will cause 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 ; **sqlcmd** defines the following additional return values:

Return Values	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 cannot 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.

If the file contains Transact-SQL statements that are not followed by **GO**, you must enter **GO** on the line that follows **:r**.

> **Note**
>
> **<** *filename* **>** is read relative to the startup directory in which **sqlcmd** was run.

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. This includes 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:

0	wait forever
n>0	wait for n seconds

The SQLCMDSERVER scripting variable will reflect the current active connection.

If *timeout* is not 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 will be prompted to enter a password. This is not true if the SQLCMDUSER or SQLCMDPASSWORD environment variables have been set. If neither options nor environment variables are provided, Windows Authentication mode is used to login. For example to connect to an instance, `instance1`, of SQL Server , `myserver`, by using integrated security you would use the following:

```
:connect myserver\instance1
```

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

```
: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 topic

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

sqlcmd File Names