sqlcmd SQL Server shell

START HERE

`sqlcmd -S <host> -U <user>`
Connect with a SQL login

`sqlcmd -S <host> -E`
Connect with integrated auth

`sqlcmd -Q "<sql>"`
Run one query and exit

`sqlcmd -i <file.sql>`
Run a script

`GO`
Send the typed batch

`:r <file>`
Pull a file into the batch

`:setvar <var> <value>`
Set a $(var)

`:quit`
Leave sqlcmd

Connect -S -U -E -d → *type / :r* → **Batch** lines, $(var) expanded → *GO* → **Server** runs the batch → *results* → **Output** screen, -o, :out

CONCEPTS

Typed lines collect in a batch; nothing runs until GO on a line of its own. GO and :commands belong to sqlcmd, not T-SQL, and must start the line.

Two variants share the name: go-sqlcmd (standalone, has sqlcmd --version and create) and ODBC sqlcmd from mssql-tools18. sqlcmd -? shows the version on both.

ODBC sqlcmd 18 encrypts by default (-Nm) and validates the certificate, so a local server with a self-signed cert needs -C.

CONNECT

`sqlcmd -S <host>,<port> -U <user>`
Connect with a SQL login
Prompts for the password if -P and SQLCMDPASSWORD are unset.

`sqlcmd -S <host>\<instance>`
Connect to a named instance

`sqlcmd -E`
Use integrated (Windows/ Kerberos) auth
Cannot be combined with -U or -P.

`SQLCMDPASSWORD=<pw> sqlcmd -U <user>`
Pass the password via environment
Safer than -P <pw>, which shows up in ps and shell history.

`sqlcmd -d <db>`
Start in a database

`sqlcmd -C`
Trust the server certificate
Needed for self-signed certs, e.g. a fresh container.

`sqlcmd -N<s|m|o>`
Encryption: strict, mandatory, optional
go-sqlcmd also takes -N true, false or disable.

`sqlcmd -G -S <srv>.database.windows.net`
Sign in with Microsoft Entra ID

`:connect <host> -U <user>`
Switch server inside a session
Not a batch separator: send pending lines with GO first.

RUN SQL

`sqlcmd -Q "<sql>"`
Run a query and exit
Separate statements with ;, never with GO.

RUN SQL (cont.)

`sqlcmd -q "<sql>"`
Run a query, then stay interactive

`sqlcmd -i <file.sql>`
Run a script file
Several files: -i a.sql,b.sql (no spaces) or repeat -i.

`sqlcmd -i <in.sql> -o <out.txt>`
Write all output to a file

`GO`
Run the buffered batch

`GO <count>`
Run the batch n times

`:r <file>`
Read a file into the batch
Relative to the start directory; follow with GO.

`:reset`
Discard the current batch

`:list`
Print the current batch

`sqlcmd -t <seconds>`
Query timeout (default none)

`:quit`
Leave sqlcmd
EXIT and QUIT also work without the colon.

`:help`
List sqlcmd commands

VARIABLES

`:setvar <var> "<value>"`
Define a scripting variable

`$(<var>)`
Insert a variable's value
Plain text substitution, also inside quoted strings.

`sqlcmd -v <var>="<value>"`
Set a variable from the shell
Missing in ODBC sqlcmd on Linux and macOS; go-sqlcmd has it.

`:listvar`
Show all variables

Built-in variables

Variable	Set by
SQLCMDSERVER	-S, :connect
SQLCMDUSER	-U
SQLCMDPASSWORD	-P
SQLCMDDBNAME	-d
SQLCMDINI	startup script path
SQLCMDEDITOR	editor for :ed

Same name in several places: :setvar wins over -v, which wins over environment variables.

OUTPUT

`sqlcmd -W`
Trim trailing spaces from columns

OUTPUT (cont.)

`sqlcmd -s "<sep>"`
Set the column separator
Pair with -W for delimited export.

`sqlcmd -h -1`
Omit column headers

`sqlcmd -w <width>`
Line width (default 80)

`sqlcmd -y <n>`
Max width of (max)/xml values
Default 256; 0 means unlimited. Not combinable with -W.

`SET NOCOUNT ON;`
Hide "(n rows affected)"

`:out <file>`
Send results to a file
:out stdout switches back.

`sqlcmd -r1`
Messages to stderr
-r0 moves only errors of severity 11 and up.

`sqlcmd --vertical`
One column per line (go-sqlcmd)

ERRORS & EXIT CODES

`sqlcmd -b`
Stop and exit 1 on an error
Errors above severity 10 count; check $? in scripts.

`:on error exit`
Stop the script on an error

`sqlcmd -V <severity>`
Minimum severity that sets the exit code
Docs suggest -V 16 for automated quality gates.

`:exit(<query>)`
Exit with a query result as code
Linux and macOS pass only the low byte.

GO-SQLCMD CONTAINERS

`sqlcmd create mssql --accept-eula`
Start SQL Server in a container
Needs Docker or Podman; adds a context to ~/.sqlcmd/sqlconfig.

`sqlcmd create mssql --accept-eula --using <url.bak>`
Same, restoring a backup

`sqlcmd create mssql --tag <tag>`
Pick a SQL Server version

`sqlcmd query "<sql>"`
Query the current context

`sqlcmd config get-contexts`
List contexts

GO-SQLCMD CONTAINERS (cont.)

`sqlcmd config use-context <name>`
Switch context

`sqlcmd config connection-strings`
ODBC, ADO.NET, JDBC strings

`sqlcmd stop / sqlcmd start`
Stop or start the container

`sqlcmd delete`
Remove container and context
--force skips the safety checks.

T-SQL HELPERS

`SELECT name FROM sys.databases;`
List databases

`USE <db>;`
Switch database

`SELECT name FROM sys.tables;`
List tables in this database

`EXEC sp_help '<table>';`
Describe a table

`SELECT TOP (10) * FROM <table>;`
First rows (no LIMIT in T-SQL)

`SELECT @@VERSION;`
Server version

`SELECT DB_NAME(), SUSER_SNAME();`
Current database and login

RECIPES

Throwaway dev server

```
sqlcmd create mssql --accept-eula --using https://aka.ms/AdventureWorksLT.bak
sqlcmd query "SELECT name FROM sys.databases"
sqlcmd config connection-strings
sqlcmd delete --force
```

Script in CI, fail on error

```
export SQLCMDPASSWORD="$DB_PASSWORD"
sqlcmd -S <host> -U <user> -d <db> -b -V 16 -i migrate.sql
# non-zero exit code stops the pipeline
```

Export a query as CSV

```
sqlcmd -S <host> -E -d <db> -W -s "," -h -1 \
  -Q "SET NOCOUNT ON; SELECT id, name FROM <table>" -o out.csv
```

Parameterised deploy script

```
-- deploy.sql
CREATE DATABASE [$(Db)];
GO
USE [$(Db)];
:r ./schema.sql
GO
# run: sqlcmd -S <host> -U <user> -v Db=AppDb -i deploy.sql
```