Cypher Shell Neo4j graph CLI

START HERE

`cypher-shell -u neo4j -p <password>`
Open the shell

`:help`
List shell commands

`MATCH (n) RETURN n LIMIT 5;`
Peek at some nodes

`:exit`
Leave the shell

`CREATE (:Person {name: 'Ann'});`
Add a node

`MATCH (a)-[r]→(b) RETURN a, r, b;`
Follow relationships

`SHOW DATABASES;`
List databases

`:use <db>`
Switch database

cypher-shell your terminal → Bolt → **Neo4j server** port 7687 → routes to → **Database** neo4j / system → holds → **Graph** nodes + relationships

CONCEPTS

Statements typed in the shell must end with a semicolon; lines starting with : are shell commands and need none.

Patterns are drawn in ASCII art: (node)-[:REL]→(node). Labels start with a colon, properties go in braces.

CONNECT

`cypher-shell -u <user> -p <password>`
Connect to localhost
Defaults to neo4j://localhost:7687. Leave -p out to be prompted.

`cypher-shell -a <uri>`
Connect to another server
Schemes such as neo4j+s:// enable TLS. Env: NEO4J_URI.

`cypher-shell -d <database>`
Start in a given database

`cypher-shell --access-mode read`
Open a read-only session

`cypher-shell -v`
Print the shell version

Environment variables

Variable	Replaces
`NEO4J_USERNAME`	-u
`NEO4J_PASSWORD`	-p
`NEO4J_ADDRESS`	-a
`NEO4J_DATABASE`	-d

SHELL COMMANDS

`:help`
List shell commands
Use :help <command> for one command.

`:exit`
Quit the shell

`:use <database>`
Set the active database

`:begin`
Open an explicit transaction

`:commit`
Commit the open transaction

`:rollback`
Roll back the open transaction

`:param { alias: 'Robin' }`
Set a query parameter
Reference it in Cypher as $alias. :param alone lists them.

`:source <file>`
Run statements from a file

`:history`
Show earlier statements

`:sysinfo`
Show system information

Arrows browse history, Tab autocompletes commands, Home and End jump along the line.

SCRIPT & OUTPUT

`cypher-shell "<cypher>"`
Run one statement and exit

SCRIPT & OUTPUT (cont.)

`cypher-shell -f <file>`
Run a Cypher file and exit

`cypher-shell --format plain`
Minimal output for scripts

`cypher-shell --format verbose`
Tables plus statistics

`cypher-shell -P '{n: 3}'`
Set a session parameter
Repeat -P for several parameters.

`cypher-shell --fail-at-end`
Report errors after the whole file
The default, --fail-fast, stops at the first error.

READ

`MATCH (n) RETURN n LIMIT <k>`
Sample some nodes

`MATCH (n:<Label>) RETURN n`
Nodes with a label

`MATCH (n:<Label> {<prop>: <value>}) RETURN n`
Match by property

`MATCH (a)-[r:<TYPE>]→(b) RETURN a, r, b`
Follow a relationship type

`WHERE n.<prop> > <value>`
Filter matches
Also supports AND, OR, NOT, IN, IS NULL and STARTS WITH.

READ (cont.)

`RETURN n.<prop> AS <alias>`
Rename a returned value

`RETURN DISTINCT n.<prop>`
Remove duplicate rows

`ORDER BY n.<prop> DESC`
Sort the rows

`SKIP <n> LIMIT <m>`
Page through results

`OPTIONAL MATCH (n)⟶(m)`
Match, or give null

`MATCH (a)-[*1..3]→(b)`
Paths of one to three hops

`count(n), sum(n.<prop>), collect(n)`
Aggregate rows
Non-aggregated RETURN columns become the grouping keys.

WRITE

`CREATE (:<Label> {<prop>: <value>})`
Create a node

`CREATE (a)-[:<TYPE>]→(b)`
Create a relationship
MATCH both nodes first, then CREATE between them.

`MERGE (n:<Label> {<prop>: <value>})`
Create only if missing
Match-or-create; avoids duplicates on reruns.

`SET n.<prop> = <value>`
Set a property

WRITE (cont.)

`SET n += {<prop>: <value>}`
Update several properties

`REMOVE n.<prop>`
Remove a property

`DELETE r`
Delete a relationship

`DETACH DELETE n`
Delete a node and its relationships
Plain DELETE fails while relationships remain.

`ON CREATE SET n.<prop> = <v>`
Set only when MERGE creates

PIPELINES

`WITH n, count(*) AS c`
Pass results to the next part

`UNWIND <list> AS x`
Turn a list into rows

`UNWIND $rows AS row`
Loop over a parameter list

`LOAD CSV WITH HEADERS FROM '<url>' AS row`
Import rows from CSV
file:/// URLs read from the server's import directory.

`RETURN $name`
Use a parameter

SCHEMA & ADMIN

`SHOW DATABASES`
List databases

SCHEMA & ADMIN (cont.)

`SHOW INDEXES`
List indexes

`SHOW CONSTRAINTS`
List constraints

`CREATE DATABASE <name>`
Create a database
Enterprise Edition only; Community has one user database.

`CREATE INDEX FOR (n:<Label>) ON (n.<prop>)`
Index a property

`CREATE CONSTRAINT FOR (n:<Label>) REQUIRE n.<prop> IS UNIQUE`
Enforce unique values

`CALL db.labels()`
List node labels

`CALL db.relationshipTypes()`
List relationship types

TUNING

`EXPLAIN <query>`
Show the plan without running

`PROFILE <query>`
Run and show rows and db hits
Watch for AllNodesScan and NodeByLabelScan on big graphs.

Start from an index lookup; every MATCH without a label or indexed property scans the whole graph.