Node.js JavaScript runtime CLI

START HERE

`node <file>`
Run a script

`node --watch <file>`
Restart on change

`node`
Open the REPL

`node --test`
Run the tests

`node -e "<code>"`
Run a one-liner

`node --env-file=.env <file>`
Load a .env file

`node -p "<expr>"`
Evaluate and print

`node --inspect-brk <file>`
Debug in DevTools

NODE_OPTIONS environment defaults → **Node flags** node --watch ... → **Script** file.js or -e → **Arguments** process.argv

CONCEPTS

node runs a file as a CommonJS script, or as an ES module for .mjs, "type": "module" or --input-type=module.

Flags go before the script name; everything after it is passed to the script. Use -- to end node's own options.

Command-line flags win over NODE_OPTIONS for single-value options. Some flags (like -p) are not allowed in NODE_OPTIONS.

RUN CODE

`node <file>`
Run a script

`node`
Start the interactive REPL

`node -e "<code>"`
Evaluate a string of code

`node -p "<expr>"`
Evaluate and print the result

`node <file> <args>`
Pass arguments to the script
Read them from process.argv.slice(2).

`node -`
Read the script from stdin

`node -c <file>`
Syntax check without running

`node --input-type=module -e "<code>"`
Evaluate code as an ES module
Allows import and top-level await in -e.

`node -v`
Print the Node version

`corepack enable`
Activate pnpm and yarn shims
Bundled with Node 20 and 22; not with Node 25+.

`node -p "process.versions"`
Show V8 and library versions

WATCH MODE

`node --watch <file>`
Restart when imported files change

WATCH MODE (cont.)

`node --watch-path=<dir> <file>`
Watch specific paths instead
Implies --watch. Recursive watching is only supported on macOS and Windows.

`node --watch-preserve-output <file>`
Keep the console between restarts

TEST RUNNER

`node --test`
Run all discovered test files
*Finds *.test.js, test/**, and similar patterns.*

`node --test <file>`
Run specific test files

`node --test-name-pattern="<regex>"`
Run tests whose name matches

`node --test-only`
Run only tests marked only

`node --test-reporter=<name>`
Choose output: spec, tap, dot, junit

`node --test --watch`
Re-run tests on change

Minimal test file

```
import test from 'node:test';
import assert from 'node:assert/
strict';

test('adds', () => {
  assert.equal(1 + 1, 2);
});
```

ENVIRONMENT AND CONFIG

`node --env-file=<file> <script>`
Load variables from a .env file
Repeat the flag to load several files; later ones win.

`NODE_OPTIONS="<flags>" node <script>`
Set default flags via environment

`NODE_ENV=production node <script>`
Set a variable for one run
Only a convention that libraries read; Node itself ignores it.

ENVIRONMENT AND CONFIG (cont.)

`node --require <module>`
Preload a CommonJS module

`node --import <module>`
Preload an ES module

`node --run <script>`
Run a package.json script
Node 22+. Faster than npm run but has no lifecycle hooks.

`node --title=<name> <script>`
Set the process title

DEBUGGING

`node --inspect <file>`
Start with the inspector open
Then open chrome://inspect or attach an editor on port 9229.

`node --inspect-brk <file>`
Break on the first line

`node inspect <file>`
Use the built-in CLI debugger

`node --enable-source-maps <file>`
Map stack traces to source files

`node --trace-warnings <file>`
Show where warnings come from

`node --trace-deprecation <file>`
Show where deprecated APIs are used

`node --no-warnings <file>`
Silence all process warnings

`NODE_DEBUG=<module> node <file>`
Log internals, e.g. http, fs

PERFORMANCE AND MEMORY

`node --max-old-space-size=<MiB> <file>`
Raise the heap limit
Fixes JavaScript heap out of memory crashes.

`node --v8-options`
List all V8 flags

PERMISSIONS AND TYPESCRIPT

`node --permission --allow-fs-read=<path> <file>`
Restrict file access
Flag is --experimental-permission before Node 22.13 and 23.5.

`node <file>.ts`
Run TypeScript by stripping types
Default in Node 22.18+ and 23.6+; use --experimental-strip-types on 22.6+. No enums or paths.

`node --no-strip-types`
Turn off TypeScript support

REPL

Dot commands

Command	Effect
`.help`	List commands
`.editor`	Multi-line editor, Ctrl+D runs
`.load <file>`	Run a file in the session
`.save <file>`	Save the session to a file
`.break`	Abandon a multi-line input
`.exit`	Leave (or Ctrl+D)

Tab
Complete names and properties

_
Last evaluated result

Core modules like fs and path are available in the REPL and in -p/-e without require.

RECIPES

One-liners

```
# random UUID
node -p "crypto.randomUUID()"
# today as ISO date
node -p "new
Date().toISOString()"
# usable CPU count
node -p
"os.availableParallelism()"
```

Dev loop with env and watch

```
node --watch --env-file=.env
server.js
# tests alongside
node --test --watch
```

Debug a failing test

```
node --inspect-brk --test --
test-name-pattern="adds" test/
a.test.js
# then open chrome://inspect
```