



# Node.js I - Getting Started

# Chesapeake Node.js User Group (CNUG)



<https://www.meetup.com/Chesapeake-Region-nodeJS-Developers-Group>

# Agenda

## ➤ **Installing Node.js**

- ✓ **Background**
- ✓ **Node.js Run-time Architecture**
- ✓ **Node.js & npm software installation**
- ✓ **JavaScript Code Editors**
- ✓ **Installation verification**
- ✓ **Node.js Command Line Interface (CLI)**
- ✓ **Read-Evaluate-Print-Loop (REPL) Interactive Console**
- ✓ **Debugging Mode**
- ✓ **JSHint**
- ✓ **Documentation**

# Node.js - Background

## ➤ What is Node.js?

- ❑ Node.js is a server side (Back-end) JavaScript runtime
- ❑ Node.js runs “V8”
  - ✓ Google’s high performance JavaScript engine
  - ✓ Same engine used for JavaScript in the Chrome browser
  - ✓ Written in C++
  - ✓ <https://developers.google.com/v8/>
- ❑ Node.js implements ECMAScript
  - ✓ Specified by the **ECMA-262** specification
  - ✓ Node.js support for ECMA-262 standard by version:
    - <https://node.green/>

# Node.js - Node.js Run-time Architectural Concepts

## ➤ Node.js is designed for Single Threading

### ❑ Main Event listeners are single threaded

- ✓ Events immediately handed off to the thread pool
- ✓ This makes Node.js perfect for Containers

### ❑ JS programs are single threaded

- ✓ Use asynchronous (Non-blocking) calls

### ❑ Background worker threads for I/O

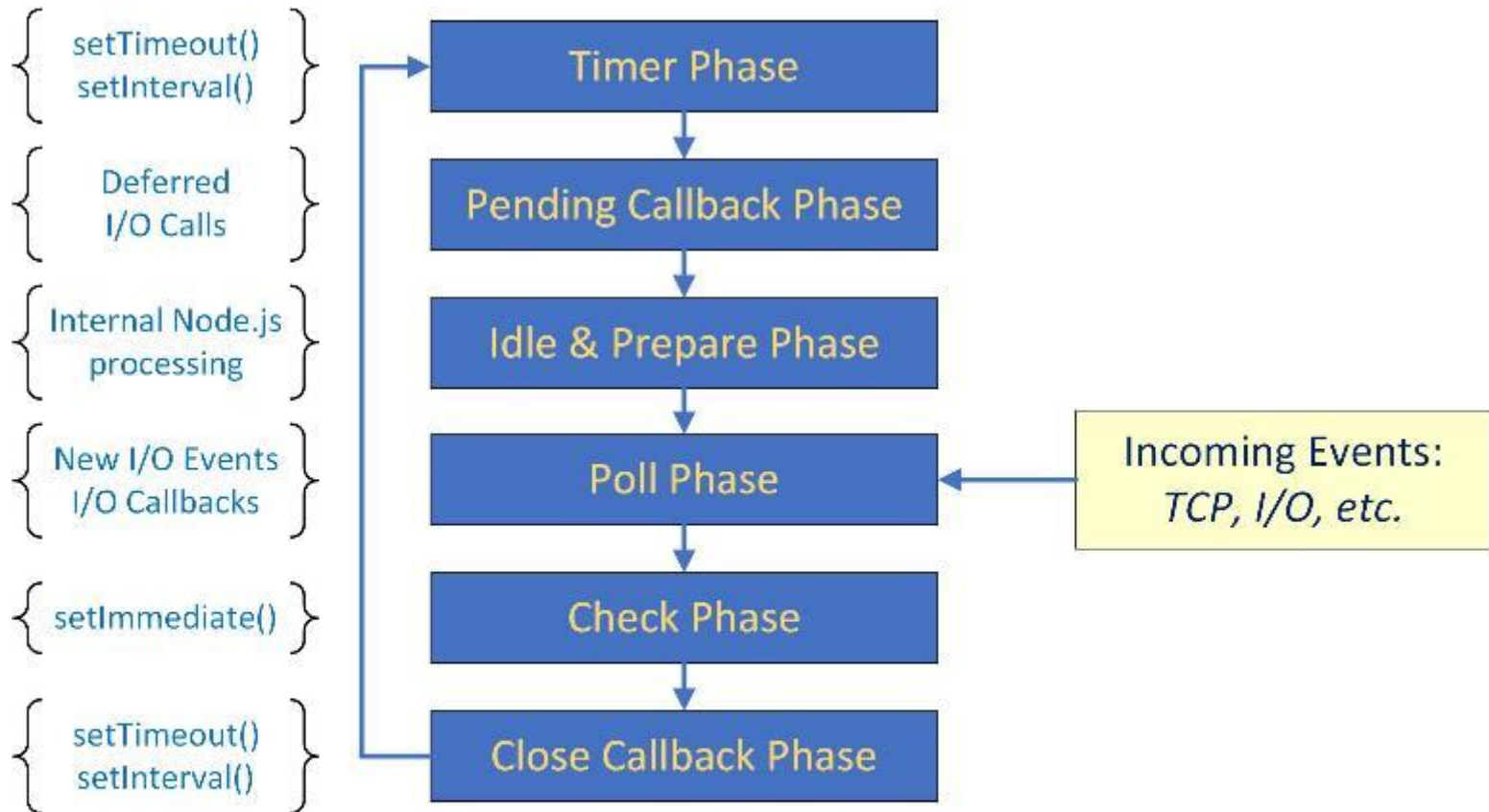
### ❑ Underlying Linux kernel is multi-threaded

## ➤ Event Loop leverages Linux multi-threading

### ❑ Events queued

### ❑ Queues processed in Round Robin fashion

# Node.js - Event Processing Loop



{ Each phase has a queue of Callbacks to process. Queue processed until empty or maximum number of Callbacks is reached. }

# Node.js - Downloading Software

## ➤ Download software from Node.js site:

<https://nodejs.org/en/download/>

### Supported Platforms

- ✓ Windows (Installer or Binary)
- ✓ Mac (Installer or Binary)
- ✓ Linux
- ✓ SunOS, Docker, IBM Power (AIX, Linux), IBM System z

### Long Term Support (LTS)

- ✓ Most stable version (8.11)

### Current

- ✓ Newest features (10.7)

# Node.js - Software Installation

## ➤ Windows – Follow installer prompts

- Standard Windows installation

  - ✓ Verified for Windows 10

- Make sure to select “npm” if not already installed

- Reboot computer



# JavaScript Code Editors

## ➤ Atom (Open Source)

❑ <https://atom.io/>

## ➤ Eclipse (IBM)

❑ <http://www.eclipse.org/downloads/packages/>

❑ Help → Eclipse Marketplace ... → Find → “node”

✓ **Nodeclipse** (1.0.2)

## ➤ Visual Studio (Microsoft)

❑ <https://code.visualstudio.com/docs/nodejs/nodejs-tutorial>

## Node.js & npm - Installation Verification

- **Launch a DOS Command Window**
  - ❑ Windows → Run → cmd.exe
  
- **Enter the following commands:**
  - ❑ **node -v** (Verify Node.js installed & in Path)
  - ❑ **npm -v** (Verify npm installed & in Path)

## Node.js - Command Line Interface (CLI)

- Windows “.bat” file to launch Command window
  - ❑ **”c:\Program Files\nodejs\nodevars.bat”**
- Execute JavaScript program
  - ❑ **node *programName.js***
    - ✓ “.js” filename extension optional (assumed if not provided)
- Generate JSDoc from comments
  - ❑ **jsdoc *programName.js***
  - ❑ HTML pages stored in **”./out”** directory

# “node” Command (1): Syntax & Options

## ➤ “node” Program syntax

- ❑ **node** *options* *programName.js* **--** *scriptArguments*
  - ✓ “.js” is assumed if no file extension is provided

## ➤ “node” Program Options

- ❑ Default option is to execute the specified program.
- ❑ Other options include:

- ✓ **-c** | **--check** ... *script* | *script.js* (Syntax check program; no execution)
- ✓ **-e** | **--eval** ... *expression* (Execute JS expression)
- ✓ **-h** | **--help** (Command syntax help)
- ✓ **-i** | **--interactive** (Enter interactive programming mode)
- ✓ **inspect** (Execute program in debugging mode)
- ✓ **-p** | **--print** ... *expression* (Print JS expression result)
- ✓ **-r** | **--require** ... *module* (Load JavaScript Module)
- ✓ **-v** | **--version** (Display Node.js version)

## “node” Command (2): Arguments

### ➤ “node” Command Arguments

☐ --

☐ The “Double Dash” follows the script name

☐ Arguments follow the “double dash”

☐ Arguments are passed to the script

✓ They do NOT affect the execution of the “node” command

## “node ” Command (3): Environment Variables

### ➤ Environment variables affect command execution

#### ❑ NODE\_PATH

- ✓ A colon (“:”) separated list of directories to search for scripts.

#### ❑ NODE\_NO\_WARNINGS

- ✓ A value of “1” suppresses warning messages.

#### ❑ NODE\_REPL\_HISTORY

- ✓ Path to the file storing REPL history (`.node_repl_history`).

#### ❑ NODE\_EXTRA\_CA\_CERTS

- ✓ File containing additional CA Signer certificates in PEM format.

#### ❑ OPENSSSL\_CONF

- ✓ Load an OpenSSL configuration file on startup.

#### ❑ SSL\_CERT\_FILE

- ✓ Specify the x.509 certificate Keystore directory.

#### ❑ SSL\_CERT\_FILE

- ✓ Specify the x.509 certificate Keystore filename.

## “node” command (4): Object Information

### ➤ Display node.js Object Information

❑ **node -p object** (Display JSON for Object)

❑ <https://nodejs.org/api/globals.html>

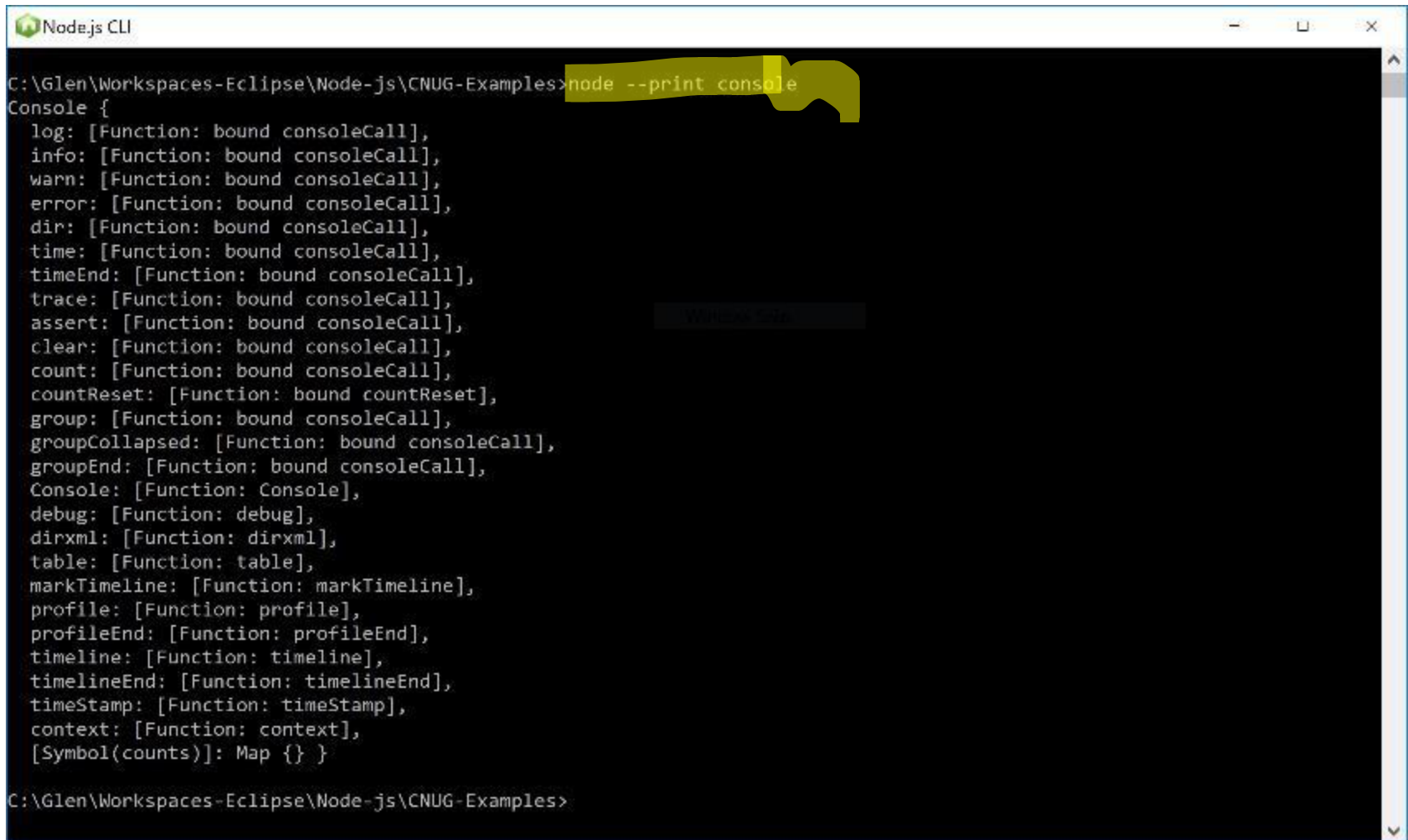
### ➤ Global Objects

- ✓ **console** (*Provides a console for **stdout**, **stderr***)
- ✓ **global** (*Global Namespace Object*)
- ✓ **os** (*Operating System Object*)
- ✓ **process** (*Current Node.js process Object*)

### ➤ Module Objects

- ✓ **module** (*Current Module Object*)
- ✓ **require** (*Current required Modules*)

# “node --print” command example



```
Node.js CLI
C:\Glen\Workspaces-Eclipse\Node-js\CNUG-Examples>node --print console
Console {
  log: [Function: bound consoleCall],
  info: [Function: bound consoleCall],
  warn: [Function: bound consoleCall],
  error: [Function: bound consoleCall],
  dir: [Function: bound consoleCall],
  time: [Function: bound consoleCall],
  timeEnd: [Function: bound consoleCall],
  trace: [Function: bound consoleCall],
  assert: [Function: bound consoleCall],
  clear: [Function: bound consoleCall],
  count: [Function: bound consoleCall],
  countReset: [Function: bound countReset],
  group: [Function: bound consoleCall],
  groupCollapsed: [Function: bound consoleCall],
  groupEnd: [Function: bound consoleCall],
  Console: [Function: Console],
  debug: [Function: debug],
  dirxml: [Function: dirxml],
  table: [Function: table],
  markTimeline: [Function: markTimeline],
  profile: [Function: profile],
  profileEnd: [Function: profileEnd],
  timeline: [Function: timeline],
  timelineEnd: [Function: timelineEnd],
  timeStamp: [Function: timeStamp],
  context: [Function: context],
  [Symbol(counts)]: Map {} }
C:\Glen\Workspaces-Eclipse\Node-js\CNUG-Examples>
```



# Node.js - “Read” - “Eval” - “Print” - “Loop” (REPL)

## ➤ “node” Interactive Mode

- ❑ REPL (pronounced “repple”)

- ❑ Invoked by:

  - ✓ **node**

  - ✓ **node -i | -interactive**

- ❑ Process input lines:

  - ✓ “R”ead JavaScript statement

  - ✓ “E”valuate and execute statement

  - ✓ “P”rint the result of the statement

  - ✓ “L”oop through subsequent input lines

    - Terminate REPL with **CTRL-C** (twice) or **CTRL-D** (once)

- ❑ Special syntax:

  - ✓ “  ” (Underline) represents previous expression

# Node.js - Debugging Mode (1)

## ➤ Entering Debugging Mode

- Launch node using “**inspect**” option
  - ✓ `node inspect program.js`
- Add breakpoint(s) to JavaScript program
  - ✓ `debugger; statement(s)`
- Execution at breakpoints enters REPL mode

## ➤ Debugging Functionality

- Breakpoints can be set in script (“**debugger;**”)
- Breakpoints can be set during debugging (“**sb();**”)
- Variables may be “watched” (“**watch();**”)
- Commands may be executed

## Node.js - Debugging Mode (2)

### ➤ Debug Mode Commands

#### □ Define Breakpoints

- ✓ **sb()** (Set breakpoint on current line)
- ✓ **sb(*lineNumber*)** (Set a breakpoint at the specified line)
- ✓ **sb('function()')** (Set a breakpoint for start of a function)
- ✓ **sb(*script.js*, #)** (Set breakpoint at specified script line)
- ✓ **cb(*script.js*, #)** (Clear specified script breakpoint)

## Node.js - Debugging Mode (3)

### ➤ Debug Mode Commands - continued

#### ☐ Command Execution (“stepping”)

- ✓ **c, cont** *(Resume execution)*
- ✓ **kill** *(Terminate script)*
- ✓ **n, next** *(Execute next expression)*
- ✓ **o, out** *(Step out of an expression)*
- ✓ **pause** *(Pause execution)*
- ✓ **restart** *(Restart script)*
- ✓ **s, step** *(Step into expression)*

## Node.js - Debugging Mode (4)

### ➤ Debug Mode Commands - continued

#### ☐ Informational Commands

- ✓ **bt, backtrace** *(Display expression execution history)*
- ✓ **exec(expr)** *(Execute expression in current context)*
- ✓ **list(#)** *(Display the surrounding script expressions)*
- ✓ **repl** *(Enter into REPL mode from current context)*
- ✓ **scripts** *(List all loaded scripts)*
- ✓ **unwatch(expr)** *(Remove expression from watch list)*
- ✓ **version** *(Display V8 version)*
- ✓ **watch(expr)** *(Add expression to watch list)*
- ✓ **watchers** *(Display all watchers and values)*

# JavaScript “Linters”

➤ **Lint is a Code Analysis Tool**

➤ **There are several JavaScript “Linters” available**

**JSLint**

- ✓ One of the oldest JavaScript Linters
- ✓ <http://www.jshint.com/>

**JSHint**

- ✓ Successor to JSLint
- ✓ <http://jshint.com/>

**ESLint**

- ✓ Powerful but requires some configuration
- ✓ <https://github.com/eslint/eslint>

# JSHint - JavaScript Code Analysis Tool

## ➤ JSHint

### Installation

- ✓ `npm install -g jshint`

### Invocation

- ✓ `jshint scriptName`

### Configured by a `.jshintrc` file.

- ✓ Start search for file in script directory
- ✓ Works up the directory structure
- ✓ May need to modify to support ECMAScript 6
  - “esnext” : true
  - “esversion” : 6

# Node.js - Official Documentation

## ➤ JavaScript API

❑ <https://nodejs.org/en/docs/>

## ➤ “node” Command Documentation

❑ [https://nodejs.org/dist/latest-v7.x/docs/api/cli.html#cli\\_command\\_line\\_options](https://nodejs.org/dist/latest-v7.x/docs/api/cli.html#cli_command_line_options)

## ➤ Node.js Debugger

❑ <https://nodejs.org/api/debugger.html>

## ➤ Node.js Event Loop Processing

❑ <https://nodejs.org/en/docs/guides/event-loop-timers-and-nexttick/>



# Questions?





Thank  
YOU