Node.js Tutorial



Node.js tutorial provides basic and advanced concepts of Node.js. Our Node.js tutorial is designed for beginners and professionals both.

Node.js is a cross-platform environment and library for running JavaScript applications which is used to create networking and server-side applications.

Our Node.js tutorial includes all topics of Node.js such as Node.js installation on windows and linux, REPL, package manager, callbacks, event loop, os, path, query string, cryptography, debugger, URL, DNS, Net, UDP, process, child processes, buffers, streams, file systems, global objects, web modules etc.

Prerequisite

Before learning Node.js, you must have the basic knowledge of JavaScript and any programming language

# What is Node.js

Node.js is a cross-platform runtime environment and library for running JavaScript applications outside the browser. It is used for creating server-side and networking web applications. It is open source and free to use. It can be downloaded from this link <https://nodejs.org/en/>

Many of the basic modules of Node.js are written in JavaScript. Node.js is mostly used to run real-time server applications.

The definition given by its official documentation is as follows:

?Node.js is a platform built on Chrome's JavaScript runtime for easily building fast and scalable network applications. Node.js uses an event-driven, non-blocking I/O model that makes it lightweight and efficient, perfect for data-intensive real-time applications that run across distributed devices.?

Node.js also provides a rich library of various JavaScript modules to simplify the development of web applications.

1. Node.js = Runtime Environment + JavaScript Library

**Different parts of Node.js**

The following diagram specifies some important parts of Node.js:

# what is nodejsFeatures of Node.js

Following is a list of some important features of Node.js that makes it the first choice of software architects.

1. **Extremely fast:**Node.js is built on Google Chrome's V8 JavaScript Engine, so its library is very fast in code execution.
2. **I/O is Asynchronous and Event Driven:**All APIs of Node.js library are asynchronous i.e. non-blocking. So a Node.js based server never waits for an API to return data. The server moves to the next API after calling it and a notification mechanism of Events of Node.js helps the server to get a response from the previous API call. It is also a reason that it is very fast.
3. **Single threaded:**Node.js follows a single threaded model with event looping.
4. **Highly Scalable:**Node.js is highly scalable because event mechanism helps the server to respond in a non-blocking way.
5. **No buffering:**Node.js cuts down the overall processing time while uploading audio and video files. Node.js applications never buffer any data. These applications simply output the data in chunks.
6. **Open source:**Node.js has an open source community which has produced many excellent modules to add additional capabilities to Node.js applications.
7. **License:**Node.js is released under the MIT license.

Install Node.js on Windows

To install and setup an environment for Node.js, you need the following two softwares available on your computer:

1. Text Editor.
2. Node.js Binary installable

**Text Editor:**

The text editor is used to type your program. For example: Notepad is used in Windows, vim or vi can be used on Windows as well as Linux or UNIX. The name and version of the text editor can be different from operating system to operating system.

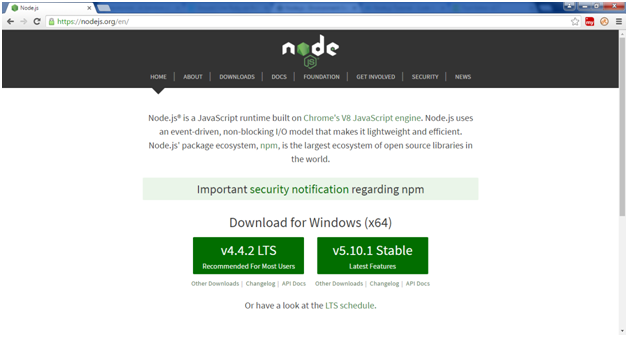
The files created with text editor are called source files and contain program source code. The source files for Node.js programs are typically named with the extension ".js".

**The Node.js Runtime:**

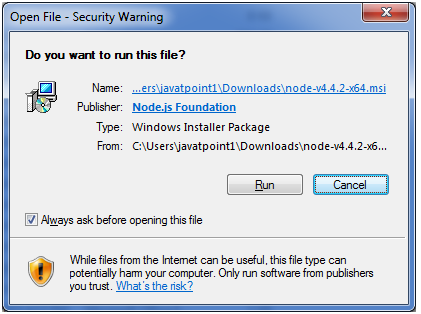
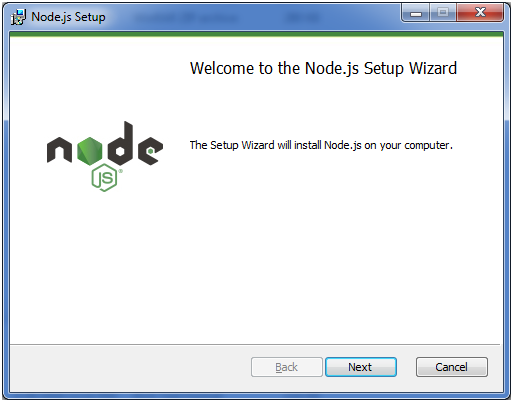
The source code written in source file is simply JavaScript. It is interpreted and executed by the Node.js interpreter.

**How to download Node.js:**

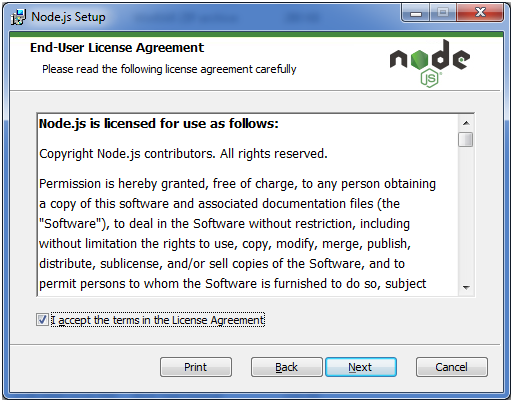
You can download the latest version of Node.js installable archive file from <https://nodejs.org/en/>



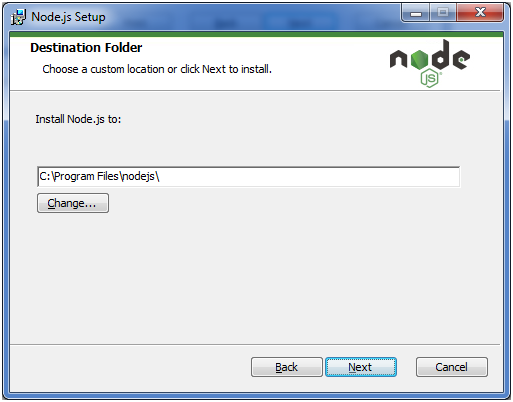
Here, you deploy the installation of node-v4.4.2 LTS recommended for most users.

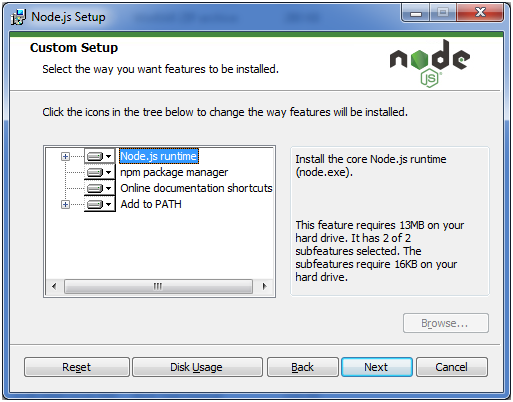
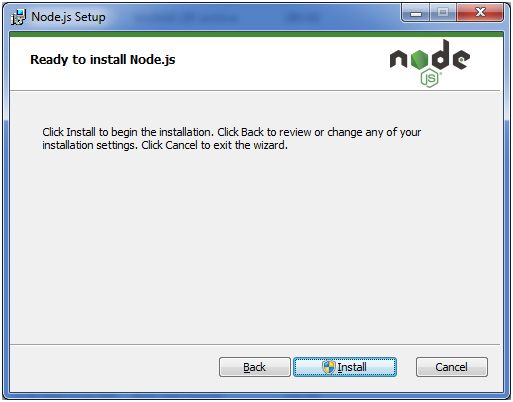
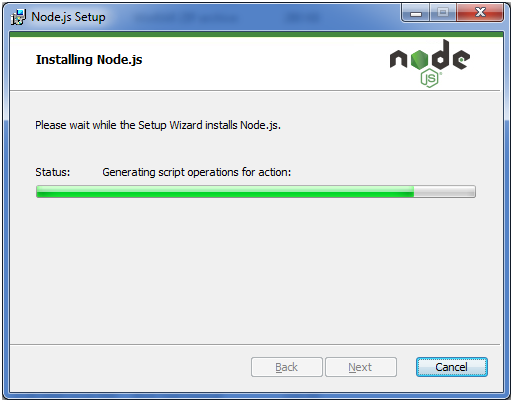
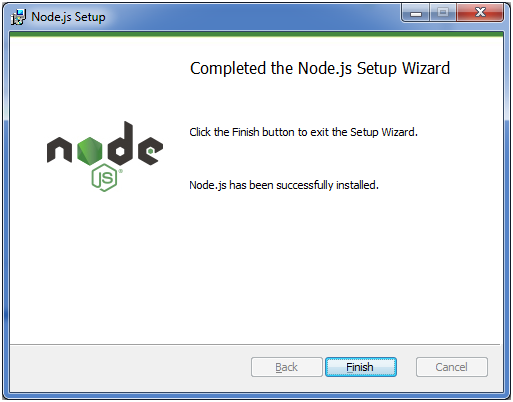
Accept the terms of license agreement.



Choose the location where you want to install.



Ready to install:

Node.js web-based Example

A node.js web application contains the following three parts:

1. **Import required modules:** The "require" directive is used to load a Node.js module.
2. **Create server:**You have to establish a server which will listen to client's request similar to Apache HTTP Server.
3. **Read request and return response:** Server created in the second step will read HTTP request made by client which can be a browser or console and return the response.

**How to create node.js web applications**

Follow these steps:

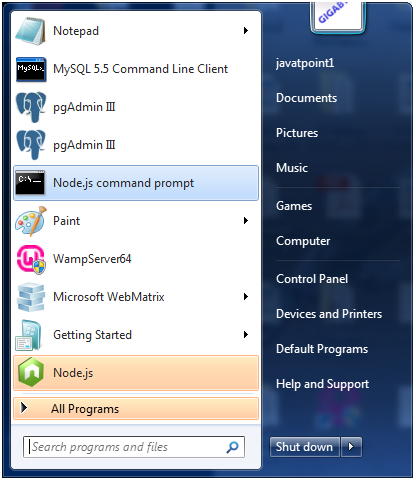
1. **Import required module:**The first step is to use ?require? directive to load http module and store returned HTTP instance into http variable. For example:
   1. var http = require("http");
2. **Create server:**In the second step, you have to use created http instance and call http.createServer() method to create server instance and then bind it at port 8081 using listen method associated with server instance. Pass it a function with request and response parameters and write the sample implementation to return "Hello World". For example:
   1. http.createServer(function (request, response) {
   2. // Send the HTTP header
   3. // HTTP Status: 200 : OK
   4. // Content Type: text/plain
   5. response.writeHead(200, {'Content-Type': 'text/plain'});
   6. // Send the response body as "Hello World"
   7. response.end('Hello World\n');
   8. }).listen(8081);
   9. // Console will print the message
   10. console.log('Server running at http://127.0.0.1:8081/');
3. **Combine step1 and step2 together** in a file named "main.js".

*File: main.js*

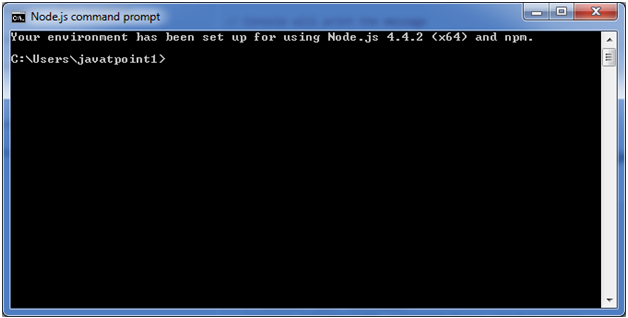
* 1. var http = require("http");
  2. http.createServer(function (request, response) {
  3. // Send the HTTP header
  4. // HTTP Status: 200 : OK
  5. // Content Type: text/plain
  6. response.writeHead(200, {'Content-Type': 'text/plain'});
  7. // Send the response body as "Hello World"
  8. response.end('Hello World\n');
  9. }).listen(8081);
  10. // Console will print the message
  11. console.log('Server running at http://127.0.0.1:8081/');

**How to start your server:**

Go to start menu and click on the Node.js command prompt.



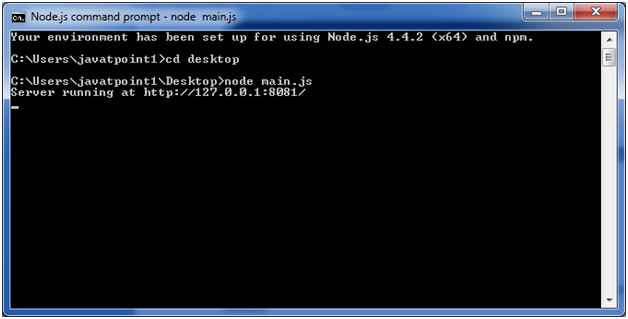
Now command prompt is open:



**Set path:**Here we have save "main.js" file on the desktop.

So type **cd desktop** on the command prompt. After that execute the main.js to start the server as follows:

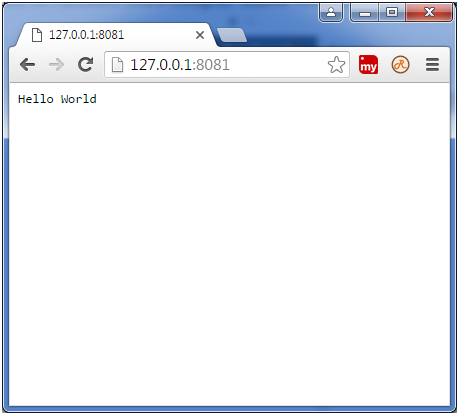
1. node main.js



Now server is started.

**Make a request to Node.js server:**

Open http://127.0.0.1:8081/ in any browser. You will see the following result.



Now, if you make any changes in the "main.js" file, you need to again run the "node main.js" command.

Node.js Console

The Node.js console module provides a simple debugging console similar to JavaScript console mechanism provided by web browsers.

There are three console methods that are used to write any node.js stream:

1. console.log()
2. console.error()
3. console.warn()

Node.js console.log()

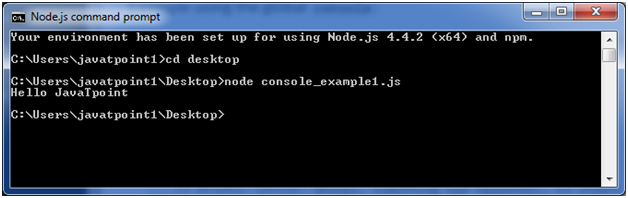
The console.log() function is used to display simple message on console.

*File: console\_example1.js*

1. console.log('Hello JavaTpoint');

Open Node.js command prompt and run the following code:

1. node console\_example1.js



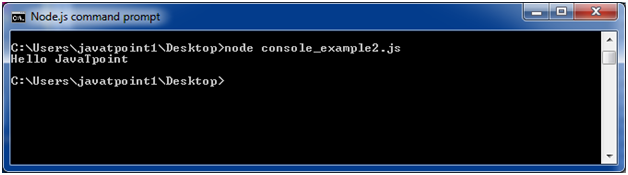
We can also use format specifier in console.log() function.

*File: console\_example2.js*

1. console.log('Hello %s', 'JavaTpoint');

Open Node.js command prompt and run the following code:

1. node console\_example2.js



Node.js console.error()

The console.error() function is used to render error message on console.

*File: console\_example3.js*

1. console.error(**new** Error('Hell! This is a wrong method.'));

Open Node.js command prompt and run the following code:

1. node console\_example3.js



Node.js console.warn()

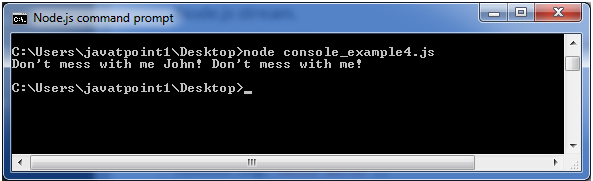
The console.warn() function is used to display warning message on console.

*File: console\_example4.js*

1. **const** name = 'John';
2. console.warn(`Don't mess with me ${name}! Don't mess with me!`);

Open Node.js command prompt and run the following code:

1. node console\_example4.js



Node.js REPL

The term REPL stands for **Read Eval Print**and**Loop**. It specifies a computer environment like a window console or a Unix/Linux shell where you can enter the commands and the system responds with an output in an interactive mode.

REPL Environment

The Node.js or node come bundled with REPL environment. Each part of the REPL environment has a specific work.

**Read:** It reads user's input; parse the input into JavaScript data-structure and stores in memory.

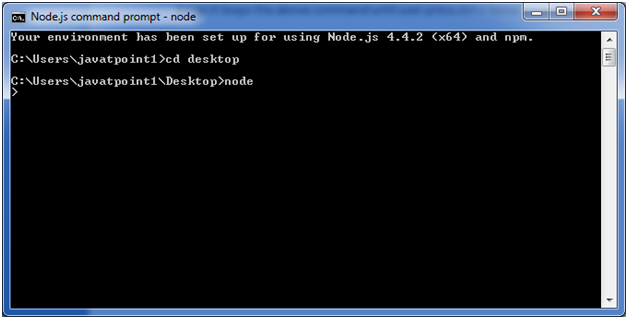
**Eval:**It takes and evaluates the data structure.

**Print:**It prints the result.

**Loop:** It loops the above command until user press ctrl-c twice.

How to start REPL

You can start REPL by simply running "node" on the command prompt. See this:

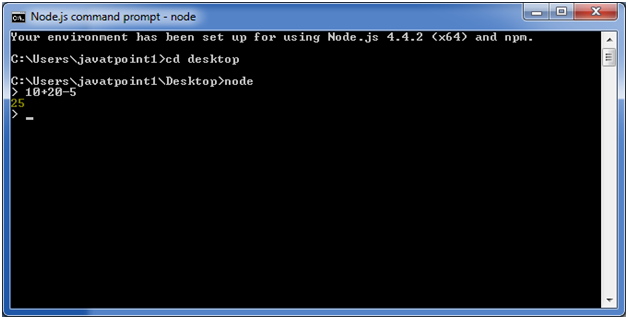


You can execute various mathematical operations on REPL Node.js command prompt:

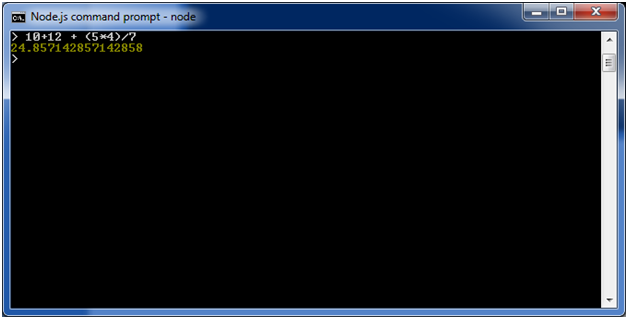
Node.js Simple expressions

After starting REPL node command prompt put any mathematical expression:

1. Example: **>**10+20-5
2. 25



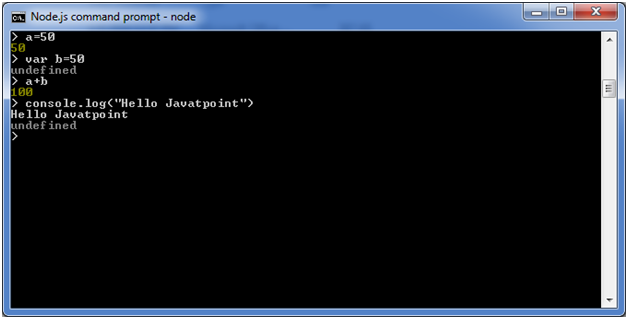
1. Example2: **>**10+12 + (5\*4)/7



Using variable

Variables are used to store values and print later. If you don't use **var**keyword then value is stored in the variable and printed whereas if **var** keyword is used then value is stored but not printed. You can print variables using console.log().

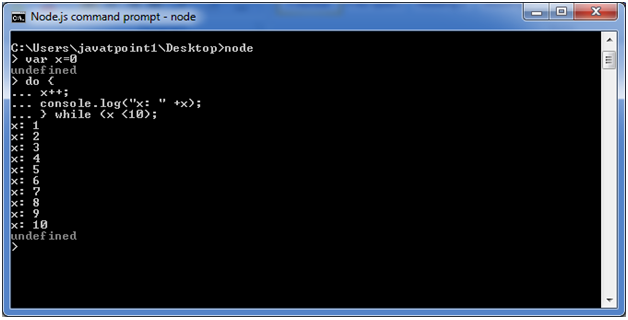
**Example:**



Node.js Multiline expressions

Node REPL supports multiline expressions like JavaScript. See the following do-while loop example:

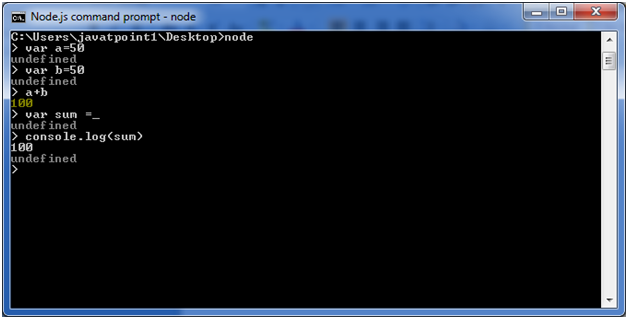
1. var x = 0
2. undefined
3. **>** do {
4. ... x++;
5. ... console.log("x: " + x);
6. ... } while ( x **<** **10** );



Node.js Underscore Variable

You can also use underscore \_ to get the last result.

**Example:**



Node.js REPL Commands

|  |  |
| --- | --- |
| Commands | Description |
| ctrl + c | It is used to terminate the current command. |
| ctrl + c twice | It terminates the node repl. |
| ctrl + d | It terminates the node repl. |
| up/down keys | It is used to see command history and modify previous commands. |
| tab keys | It specifies the list of current command. |
| .help | It specifies the list of all commands. |
| .break | It is used to exit from multi-line expressions. |
| .clear | It is used to exit from multi-line expressions. |
| .save filename | It saves current node repl session to a file. |
| .load filename | It is used to load file content in current node repl session. |

Node.js Exit REPL

Use ctrl + c command twice to come out of Node.js REPL.

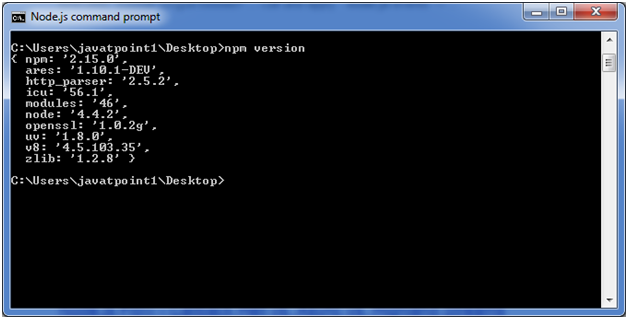
Node.js Package Manager

Node Package Manager provides two main functionalities:

* It provides online repositories for node.js packages/modules which are searchable on search.nodejs.org
* It also provides command line utility to install Node.js packages, do version management and dependency management of Node.js packages.

The npm comes bundled with Node.js installables in versions after that v0.6.3. You can check the version by opening Node.js command prompt and typing the following command:

1. npm  version



Installing Modules using npm

Following is the syntax to install any Node.js module:

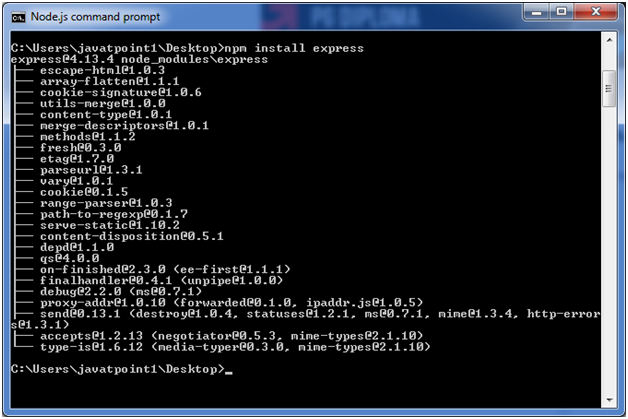
1. npm install **<Module** Name**>**

Let's install a famous Node.js web framework called express:

Open the Node.js command prompt and execute the following command:

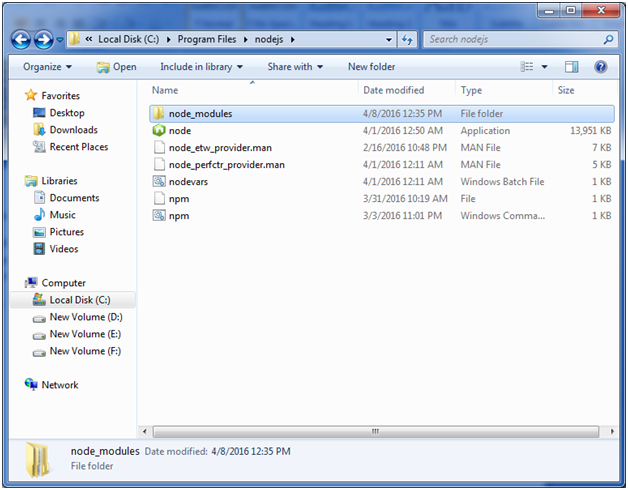
1. npm install express

You can see the result after installing the "express" framework.



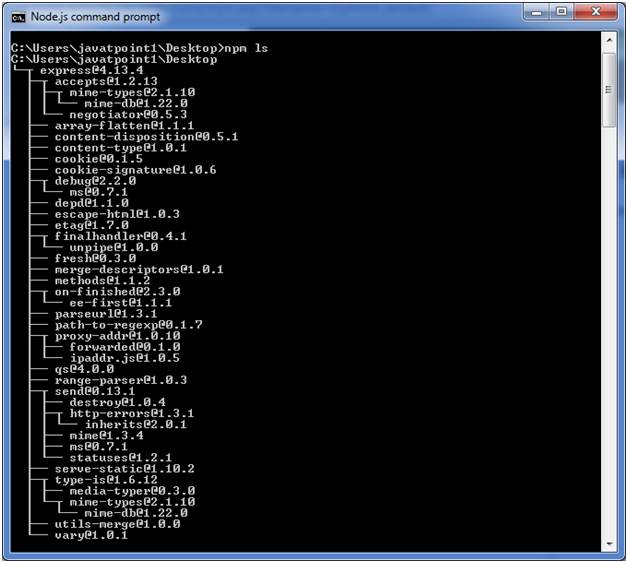
Global vs Local Installation

By default, npm installs dependency in local mode. Here local mode specifies the folder where Node application is present. For example if you installed express module, it created node\_modules directory in the current directory where it installed express module.



You can use npm ls command to list down all the locally installed modules.

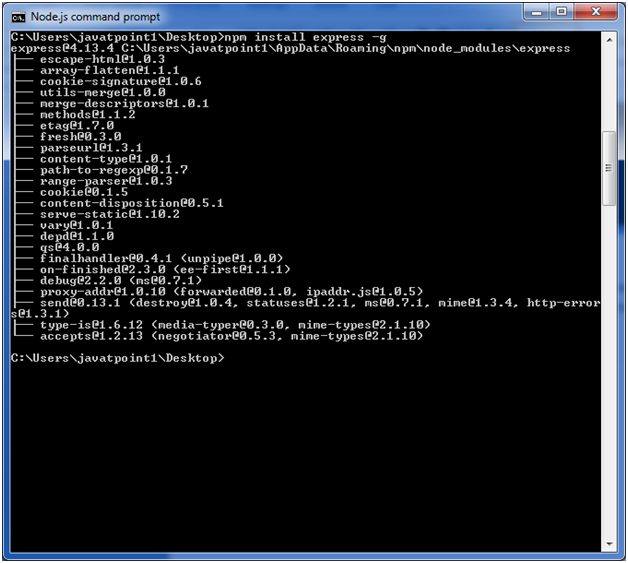
Open the Node.js command prompt and execute "npm ls":



Globally installed packages/dependencies are stored in system directory. Let's install express module using global installation. Although it will also produce the same result but modules will be installed globally.

Open Node.js command prompt and execute the following code:

1. npm install express -g

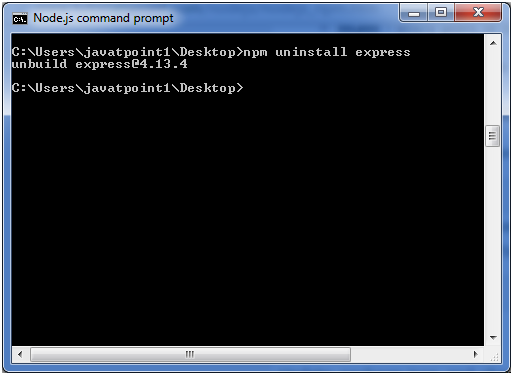


Here first line tells about the module version and its location where it is getting installed.

Uninstalling a Module

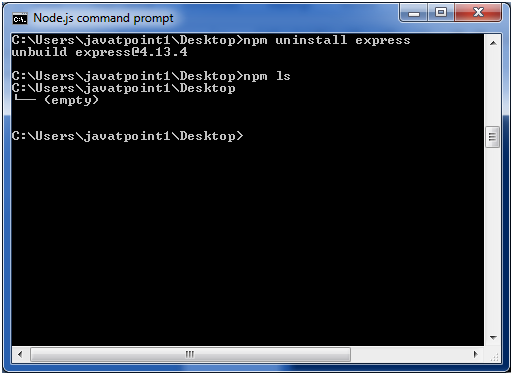
To uninstall a Node.js module, use the following command:

1. npm uninstall express



The Node.js module is uninstalled. You can verify by using the following command:

1. npm ls

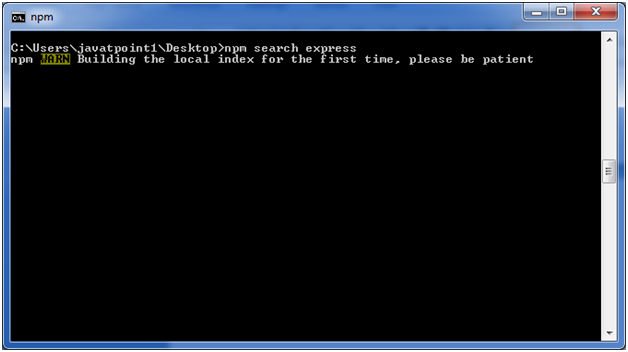
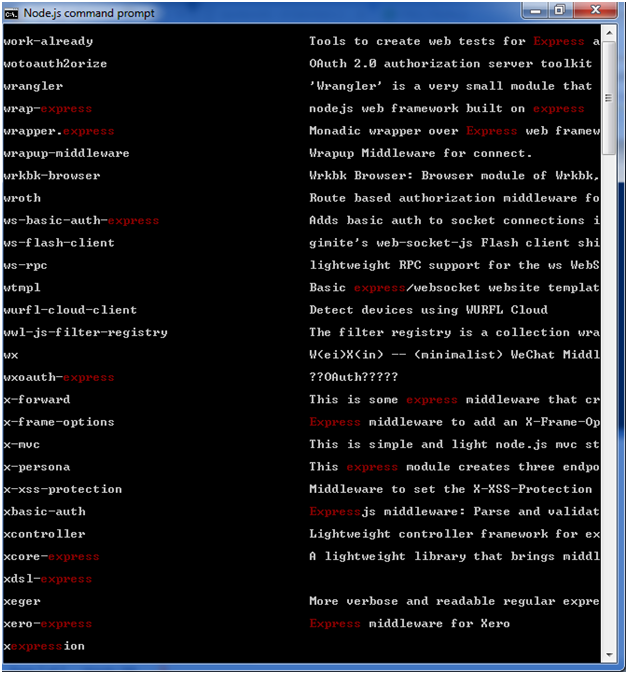


You can see that the module is empty now.

Searching a Module

"npm search express" command is used to search express or module.

1. npm search express

# Node.js Command Line Options

There is a wide variety of command line options in Node.js. These options provide multiple ways to execute scripts and other helpful run-time options.

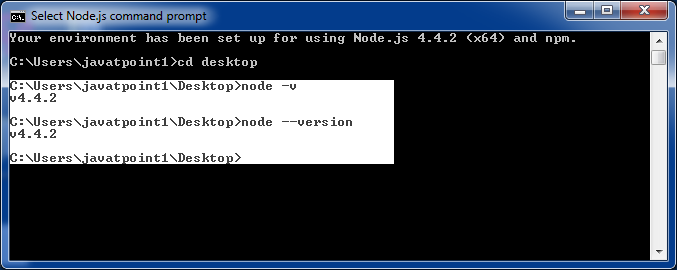
Let's see the list of Node.js command line options:

|  |  |  |
| --- | --- | --- |
| **Index** | **Option** | **Description** |
| 1. | v, --version | It is used to print node's version. |
| 2. | -h, --help | It is used to print node command line options. |
| 3. | -e, --eval "script" | It evaluates the following argument as JavaScript. The modules which are predefined in the REPL can also be used in script. |
| 4. | -p, --print "script" | It is identical to -e but prints the result. |
| 5. | -c, --check | Syntax check the script without executing. |
| 6. | -i, --interactive | It opens the REPL even if stdin does not appear to be a terminal. |
| 7. | -r, --require module | It is used to preload the specified module at startup. It follows require()'s module resolution rules. Module may be either a path to a file, or a node module name. |
| 8. | --no-deprecation | Silence deprecation warnings. |
| 9. | --trace-deprecation | It is used to print stack traces for deprecations. |
| 10. | --throw-deprecation | It throws errors for deprecations. |
| 11. | --no-warnings | It silence all process warnings (including deprecations). |
| 12. | --trace-warnings | It prints stack traces for process warnings (including deprecations). |
| 13. | --trace-sync-io | It prints a stack trace whenever synchronous i/o is detected after the first turn of the event loop. |
| 14. | --zero-fill-buffers | Automatically zero-fills all newly allocated buffer and slowbuffer instances. |
| 15. | --track-heap-objects | It tracks heap object allocations for heap snapshots. |
| 16. | --prof-process | It processes V8 profiler output generated using the v8 option --prof. |
| 17. | --V8-options | It prints V8 command line options. |
| 18. | --tls-cipher-list=list | It specifies an alternative default tls cipher list. (requires node.js to be built with crypto support. (default)) |
| 19. | --enable-fips | It enables fips-compliant crypto at startup. (requires node.js to be built with ./configure --openssl-fips) |
| 20. | --force-fips | It forces fips-compliant crypto on startup. (cannot be disabled from script code.) (same requirements as --enable-fips) |
| 21. | --icu-data-dir=file | It specifies ICU data load path. (Overrides node\_icu\_data) |

## Node.js Command Line Options Examples

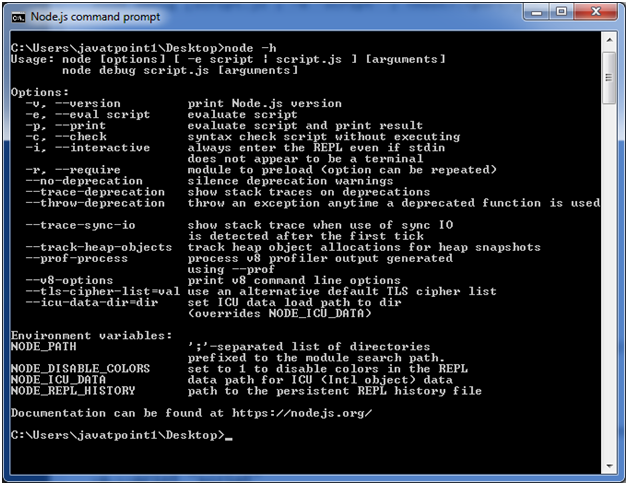
## To see the version of the running Node:

Open Node.js command prompt and run command node -v or node --version



## For Help:

Use command node ?h or node --help

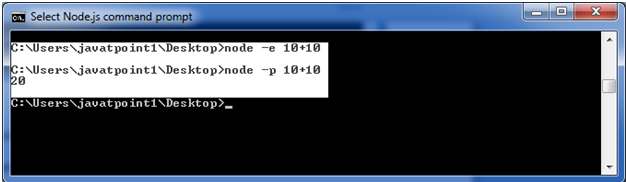


## To evaluate an argument (but not print result):

Use command node -e, --eval "script"

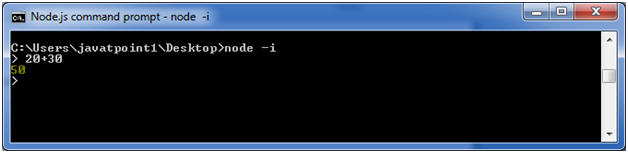
## To evaluate an argument and print result also:

Use command node -p "script"



## To open REPL even if stdin doesn't appear:

Use command node -i, or node --interactive



Node.js Global Objects

Node.js global objects are global in nature and available in all modules. You don't need to include these objects in your application; rather they can be used directly. These objects are modules, functions, strings and object etc. Some of these objects aren't actually in the global scope but in the module scope.

A list of Node.js global objects are given below:

* \_\_dirname
* \_\_filename
* Console
* Process
* Buffer
* setImmediate(callback[, arg][, ...])
* setInterval(callback, delay[, arg][, ...])
* setTimeout(callback, delay[, arg][, ...])
* clearImmediate(immediateObject)
* clearInterval(intervalObject)
* clearTimeout(timeoutObject)

Node.js \_\_dirname

It is a string. It specifies the name of the directory that currently contains the code.

*File: global-example1.js*

1. console.log(\_\_dirname);

Open Node.js command prompt and run the following code:

1. node global-example1.js



Node.js \_\_filename

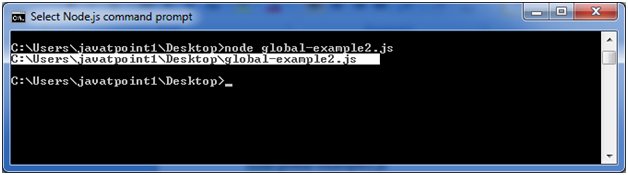
It specifies the filename of the code being executed. This is the resolved absolute path of this code file. The value inside a module is the path to that module file.

*File: global-example2.js*

1. console.log(\_\_filename);

Open Node.js command prompt and run the following code:

1. node global-example2.js



# Node.js OS

Node.js OS provides some basic operating-system related utility functions. Let's see the list generally used functions or methods.

|  |  |  |
| --- | --- | --- |
| **Index** | **Method** | **Description** |
| 1. | os.arch() | This method is used to fetch the operating system CPU architecture. |
| 2. | os.cpus() | This method is used to fetch an array of objects containing information about each cpu/core installed: model, speed (in MHz), and times (an object containing the number of milliseconds the cpu/core spent in: user, nice, sys, idle, and irq). |
| 3. | os.endianness() | This method returns the endianness of the cpu.  Its possible values are 'BE' for big endian or 'LE' for little endian. |
| 4. | os.freemem() | This methods returns the amount of free system memory in bytes. |
| 5. | os.homedir() | This method returns the home directory of the current user. |
| 6. | os.hostname() | This method is used to returns the hostname of the operating system. |
| 7. | os.loadavg() | This method returns an array containing the 1, 5, and 15 minute load averages. The load average is a time fraction taken by system activity, calculated by the operating system and expressed as a fractional number. |
| 8. | os.networkinterfaces() | This method returns a list of network interfaces. |
| 9. | os.platform() | This method returns the operating system platform of the running computer i.e.'darwin', 'win32','freebsd', 'linux', 'sunos' etc. |
| 10. | os.release() | This method returns the operating system release. |
| 11. | os.tmpdir() | This method returns the operating system's default directory for temporary files. |
| 12. | os.totalmem() | This method returns the total amount of system memory in bytes. |
| 13. | os.type() | This method returns the operating system name. For example 'linux' on linux, 'darwin' on os x and 'windows\_nt' on windows. |
| 14. | os.uptime() | This method returns the system uptime in seconds. |
| 15. | os.userinfo([options]) | This method returns a subset of the password file entry for the current effective user. |

Node.js Timer

Node.js Timer functions are global functions. You don't need to use require() function in order to use timer functions. Let's see the list of timer functions.

**Set timer functions:**

* **setImmediate():** It is used to execute setImmediate.
* **setInterval():** It is used to define a time interval.
* **setTimeout():** ()- It is used to execute a one-time callback after delay milliseconds.

**Clear timer functions:**

* **clearImmediate(immediateObject):** It is used to stop an immediateObject, as created by setImmediate
* **clearInterval(intervalObject):** It is used to stop an intervalObject, as created by setInterval
* **clearTimeout(timeoutObject):** It prevents a timeoutObject, as created by setTimeout

Node.js Timer setInterval() Example

This example will set a time interval of 1000 millisecond and the specified comment will be displayed after every 1000 millisecond until you terminate.

Node.js Errors

The Node.js applications generally face four types of errors:

* **Standard JavaScript errors** i.e. <EvalError>, <SyntaxError>, <RangeError>, <ReferenceError>, <TypeError>, <URIError> etc.
* **System errors**
* **User-specified errors**
* **Assertion errors**

Node.js DNS

The Node.js DNS module contains methods to get information of given hostname. Let's see the list of commonly used DNS functions:

* dns.getServers()
* dns.setServers(servers)
* dns.lookup(hostname[, options], callback)
* dns.lookupService(address, port, callback)
* dns.resolve(hostname[, rrtype], callback)
* dns.resolve4(hostname, callback)
* dns.resolve6(hostname, callback)
* dns.resolveCname(hostname, callback)
* dns.resolveMx(hostname, callback)
* dns.resolveNs(hostname, callback)
* dns.resolveSoa(hostname, callback)
* dns.resolveSrv(hostname, callback)
* dns.resolvePtr(hostname, callback)
* dns.resolveTxt(hostname, callback)
* dns.reverse(ip, callback)

Node.js Net

Node.js provides the ability to perform socket programming. We can create chat application or communicate client and server applications using socket programming in Node.js. The Node.js net module contains functions for creating both servers and clients.

Node.js Net Example

In this example, we are using two command prompts:

* Node.js command prompt for server.
* Window's default command prompt for client.

# Node.js Crypto

The Node.js Crypto module supports cryptography. It provides cryptographic functionality that includes a set of wrappers for open SSL's hash HMAC, cipher, decipher, sign and verify functions.

## What is Hash

A hash is a fixed-length string of bits i.e. procedurally and deterministically generated from some arbitrary block of source data.

## What is HMAC

HMAC stands for Hash-based Message Authentication Code. It is a process for applying a hash algorithm to both data and a secret key that results in a single final hash.

Node.js TLS/SSL

What is TLS/SSL

TLS stands for Transport Layer Security. It is the successor to Secure Sockets Layer (SSL). TLS along with SSL is used for cryptographic protocols to secure communication over the web.

TLS uses public-key cryptography to encrypt messages. It encrypts communication generally on the TCP layer.

What is public-key cryptography

In public-key cryptography, each client and each server has two keys: public key and private key. Public key is shared with everyone and private key is secured. To encrypt a message, a computer requires its private key and the recipient?s public key. On the other hand, to decrypt the message, the recipient requires its own

You have to use **require('tls')** to access this module.

**Syntax:**

1. var tls = require('tls');

The tls module uses OpenSSL to attain Transport Layer Security and Secure Socket Layer. TLS/SSL is a public/private key infrastructure. Each client and each server must have a private key.

A private key can be created like this:

1. openssl genrsa -out ryans-key.pem 1024

All severs and some clients need to have a certificate. Certificates are public keys signed by a Certificate Authority or self-signed. To get certificate, you have to create a "Certificate Signing Request" (CSR) file.

A certificate can be created like this:

1. openssl req -new -key ryans-key.pem -out ryans-csr.pem

To create a self-signed certificate with the CSR:

1. openssl x509 -req -in ryans-csr.pem -signkey ryans-key.pem -out ryans-cert.pem

Node.js Debugger

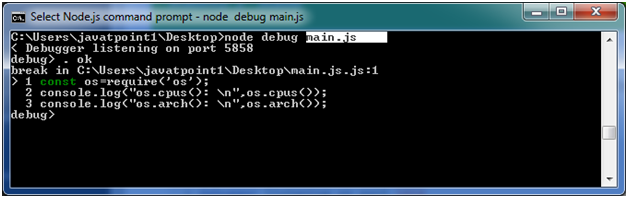
Node.js provides a simple TCP based protocol and built-in debugging client. For debugging your JavaScript file, you can use debug argument followed by the js file name you want to debug.

**Syntax:**

1. node debug [script.js | -e "script" | <host>:<port>]

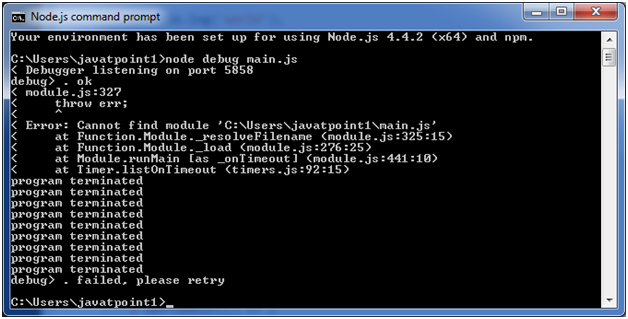
**Example:**

1. node debug main.js



**If you make any error:**

If you make any error in your js file source code or provide a wrong path on the Node.js command prompt then you will get the following result.



# Node.js Process

Node.js provides the facility to get process information such as process id, architecture, platform, version, release, uptime, upu usage etc. It can also be used to kill process, set uid, set groups, unmask etc.

The process is a global object, an instance of EventEmitter, can be accessed from anywhere.

## Node.js Process Properties

A list of commonly used Node.js process properties are given below.

|  |  |
| --- | --- |
| **Property** | **Description** |
| arch | returns process architecture: 'arm', 'ia32', or 'x64' |
| args | returns commands line arguments as an array |
| env | returns user environment |
| pid | returns process id of the process |
| platform | returns platform of the process: 'darwin', 'freebsd', 'linux', 'sunos' or 'win32' |
| release | returns the metadata for the current node release |
| version | returns the node version |
| versions | returns the node version and its dependencies |

## Node.js Process Functions

A list of commonly used Node.js process functions are given below.

|  |  |
| --- | --- |
| **Function** | **Description** |
| cwd() | returns path of current working directory |
| hrtime() | returns the current high-resolution real time in a [seconds, nanoseconds] array |
| memoryUsage() | returns an object having information of memory usage. |
| process.kill(pid[, signal]) | is used to kill the given pid. |
| uptime() | returns the Node.js process uptime in seconds. |

Node.js Child Process

The Node.js child process module provides the ability to spawn child processes in a similar manner to popen(3).

There are three major way to create child process:

* **child\_process.exec() method:** This method runs a command in a console and buffers the output.
* **child\_process.spawn() method:** This method launches a new process with a given command.
* **child\_process.fork() method:** This method is a special case of spawn() method to create child processes.

Node.js child\_process.exec() method

The child\_process.exec() method runs a command in a console and buffers the output.

**Syntax:**

1. child\_process.exec(command[, options], callback)

**Parameters:**

1) command: It specifies the command to run, with space-separated arguments.

2) options: It may contain one or more of the following options:

* **cwd:** It specifies the current working directory of the child process.
* **env:** It specifies environment key-value pairs.
* **encoding:** String (Default: 'utf8')
* **shell:** It specifies string Shell to execute the command with (Default: '/bin/sh' on UNIX, 'cmd.exe' on Windows, The shell should understand the -c switch on UNIX or /s /c on Windows. On Windows, command line parsing should be compatible with cmd.exe.)
* **timeout:** Number (Default: 0)
* **maxBuffer:** Number (Default: 200\*1024)
* **killSignal:** String (Default: 'SIGTERM')
* **uid Number:** Sets the user identity of the process.
* **gid Number:** Sets the group identity of the process.

**callback:** The callback function specifies three arguments error, stdout and stderr which is called with the following output when process terminates.

Node.js child\_process.spawn() method

The child\_process.spawn() method launches a new process with a given command. This method returns streams (stdout & stderr) and it is generally used when the process returns large amount of data.

**Syntax:**

1. child\_process.spawn(command[, args][, options])

**Parameters:**

1) command: It specifies the command to run.

2) args: It specifies an array List of string arguments.

3) options: It may contain one or more of the following options:

* **cwd:** It specifies the current working directory of the child process.
* **env:** It specifies environment key-value pairs.
* **stdio:** Array|String Child's stdio configuration
* **customFds:** Array Deprecated File descriptors for the child to use for stdio
* **detached Boolean :** The child will be a process group leader
* **uid Number:** Sets the user identity of the process.
* **gid Number:** Sets the group identity of the process

Node.js child\_process.fork() method

The child\_process.fork method is a special case of the spawn() to create Node processes. This method returns object with a built-in communication channel in addition to having all the methods in a normal ChildProcess instance.

**Syntax:**

1. child\_process.fork(modulePath[, args][, options])

**Parameters:**

1) modulePath: This is a string specifies the module to run in the child.

2) args: It specifies an array List of string arguments.

3) options: It may contain one or more of the following options:

* **cwd:** It specifies the current working directory of the child process.
* **env:** It specifies environment key-value pairs.
* **execPath:** This is a string Executable used to create the child process.
* **execArgv:** It specifies Array List of string arguments passed to the executable (Default: process.execArgv).
* **silent:** It specifies Boolean If true, stdin, stdout, and stderr of the child will be piped to the parent, otherwise they will be inherited from the parent, see the "pipe" and "inherit" options for spawn()'s stdio for more details (default is false).
* **uid Number:** Sets the user identity of the process.
* **gid Number:** Sets the group identity of the process.

Node.js Buffers

Node.js provides Buffer class to store raw data similar to an array of integers but corresponds to a raw memory allocation outside the V8 heap. Buffer class is used because pure JavaScript is not nice to binary data. So, when dealing with TCP streams or the file system, it's necessary to handle octet streams.

Buffer class is a global class. It can be accessed in application without importing buffer module.

Node.js Creating Buffers

There are many ways to construct a Node buffer. Following are the three mostly used methods:

1. **Create an uninitiated buffer:** Following is the syntax of creating an uninitiated buffer of 10 octets:
   1. var buf = new Buffer(10);
2. **Create a buffer from array:**Following is the syntax to create a Buffer from a given array:
   1. var buf = new Buffer([10, 20, 30, 40, 50]);
3. **Create a buffer from string:**Following is the syntax to create a Buffer from a given string and optionally encoding type:
   1. var buf = new Buffer("Simply Easy Learning", "utf-8");

Node.js Writing to buffers

Following is the method to write into a Node buffer:

**Syntax:**

1. buf.write(string[, offset][, length][, encoding])

**Parameter explanation:**

**string:** It specifies the string data to be written to buffer.

**offset:** It specifies the index of the buffer to start writing at. Its default value is 0.

**length:** It specifies the number of bytes to write. Defaults to buffer.length

**encoding:** Encoding to use. 'utf8' is the default encoding.

**Return values from writing buffers:**

This method is used to return number of octets written. In the case of space shortage for buffer to fit the entire string, it will write a part of the string.

**Let's take an example:**

Create a JavaScript file named "main.js" having the following code:

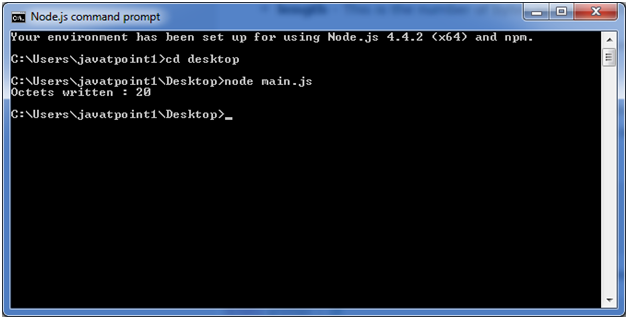
*File: main.js*

1. buf = new Buffer(256);
2. len = buf.write("Simply Easy Learning");
3. console.log("Octets written : "+  len);

Open the Node.js command prompt and execute the following code:

1. node main.js

**Output:**



Node.js Reading from buffers

Following is the method to read data from a Node buffer.

**Syntax:**

1. buf.toString([encoding][, start][, end])

**Parameter explanation:**

**encoding:** It specifies encoding to use. 'utf8' is the default encoding

**start:** It specifies beginning index to start reading, defaults to 0.

**end:**It specifies end index to end reading, defaults is complete buffer.

**Return values reading from buffers:**

This method decodes and returns a string from buffer data encoded using the specified character set encoding.

Let's take an example:

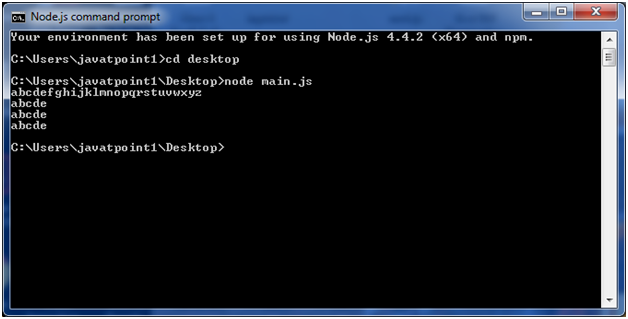
*File: main.js*

1. buf = new Buffer(26);
2. for (var i = 0 ; i **<** **26** ; i++) {
3. buf[i] = i + 97;
4. }
5. console.log( buf.toString('ascii'));       // outputs: abcdefghijklmnopqrstuvwxyz
6. console.log( buf.toString('ascii',0,5));   // outputs: abcde
7. console.log( buf.toString('utf8',0,5));    // outputs: abcde
8. console.log( buf.toString(undefined,0,5)); // encoding defaults to 'utf8', outputs abcde

Open Node.js command prompt and execute the following code:

1. node main.js

**Output:**



Node.js Streams

Streams are the objects that facilitate you to read data from a source and write data to a destination. There are four types of streams in Node.js:

* **Readable:**This stream is used for read operations.
* **Writable:**This stream is used for write operations.
* **Duplex:**This stream can be used for both read and write operations.
* **Transform:**It is type of duplex stream where the output is computed according to input.

Each type of stream is an Event emitter instance and throws several events at different times. Following are some commonly used events:

* **Data:**This event is fired when there is data available to read.
* **End:**This event is fired when there is no more data available to read.
* **Error:**This event is fired when there is any error receiving or writing data.
* **Finish:**This event is fired when all data has been flushed to underlying system.

Node.js Reading from stream

Create a text file named input.txt having the following content:

1. Javatpoint is a one of the best online tutorial website to learn different technologies in a very easy and efficient manner.

Create a JavaScript file named main.js having the following code:

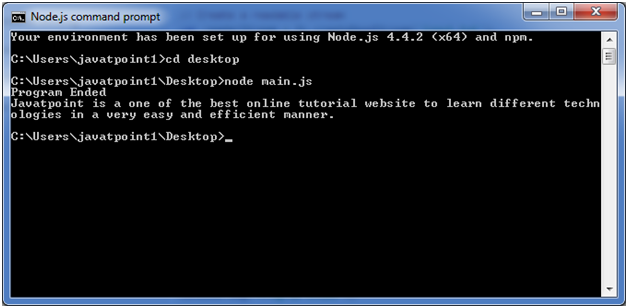
*File: main.js*

1. var fs = require("fs");
2. var data = '';
3. // Create a readable stream
4. var readerStream = fs.createReadStream('input.txt');
5. // Set the encoding to be utf8.
6. readerStream.setEncoding('UTF8');
7. // Handle stream events --**>** data, end, and error
8. readerStream.on('data', function(chunk) {
9. data += chunk;
10. });
11. readerStream.on('end',function(){
12. console.log(data);
13. });
14. readerStream.on('error', function(err){
15. console.log(err.stack);
16. });
17. console.log("Program Ended");

Now, open the Node.js command prompt and run the main.js

1. node main.js

**Output:**



Node.js Writing to stream

Create a JavaScript file named main.js having the following code:

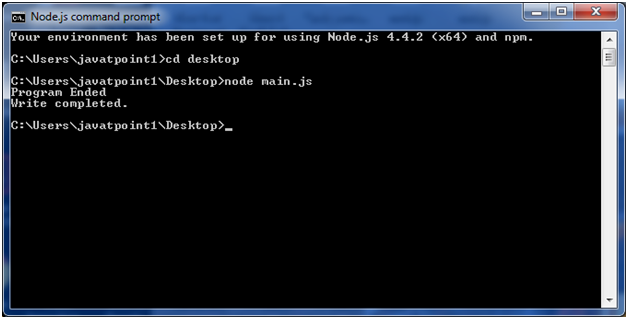
*File: main.js*

1. var fs = require("fs");
2. var data = 'A Solution of all Technology';
3. // Create a writable stream
4. var writerStream = fs.createWriteStream('output.txt');
5. // Write the data to stream with encoding to be utf8
6. writerStream.write(data,'UTF8');
7. // Mark the end of file
8. writerStream.end();
9. // Handle stream events --**>** finish, and error
10. writerStream.on('finish', function() {
11. console.log("Write completed.");
12. });
13. writerStream.on('error', function(err){
14. console.log(err.stack);
15. });
16. console.log("Program Ended");

Now open the Node.js command prompt and run the main.js

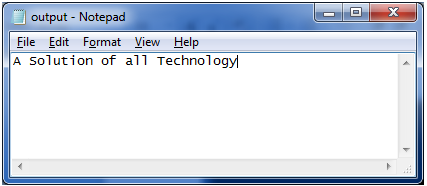
1. node main.js

You will see the following result:



Now, you can see that a text file named "output.txt" is created where you had saved "input.txt" and "main.js" file. In my case, it is on desktop.

Open the "output.txt" and you will see the following content.



Node.js Piping Streams

Piping is a mechanism where output of one stream is used as input to another stream. There is no limit on piping operation.

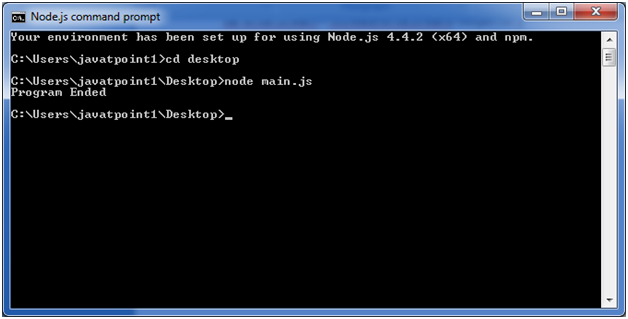
Let's take a piping example for reading from one file and writing it to another file.

*File: main.js*

1. var fs = require("fs");
2. // Create a readable stream
3. var readerStream = fs.createReadStream('input.txt');
4. // Create a writable stream
5. var writerStream = fs.createWriteStream('output.txt');
6. // Pipe the read and write operations
7. // read input.txt and write data to output.txt
8. readerStream.pipe(writerStream);
9. console.log("Program Ended");

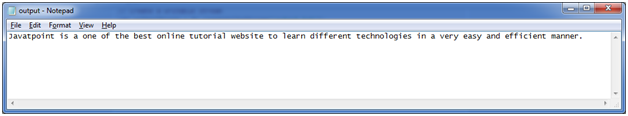
Open the Node.js and run the mian.js

1. node main.js



Now, you can see that a text file named "output.txt" is created where you had saved ?main.js? file. In my case, it is on desktop.

Open the "output.txt" and you will see the following content.



Node.js Chaining Streams

Chaining stream is a mechanism of creating a chain of multiple stream operations by connecting output of one stream to another stream. It is generally used with piping operation.

Let's take an example of piping and chaining to compress a file and then decompress the same file.

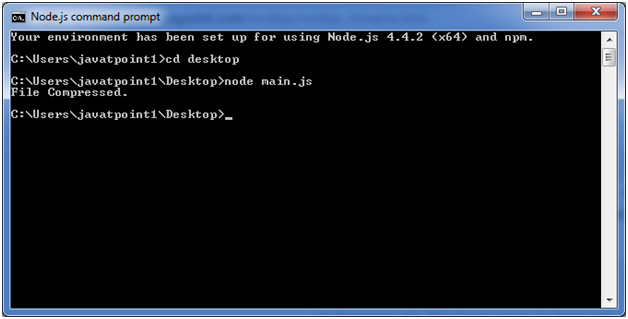
*File: main.js*

1. var fs = require("fs");
2. var zlib = require('zlib');
3. // Compress the file input.txt to input.txt.gz
4. fs.createReadStream('input.txt')
5. .pipe(zlib.createGzip())
6. .pipe(fs.createWriteStream('input.txt.gz'));
7. console.log("File Compressed.");

Open the Node.js command prompt and run main.js

1. node main.js

You will get the following result:



Now you will see that file "input.txt" is compressed and a new file is created named "input.txt.gz" in the current file.

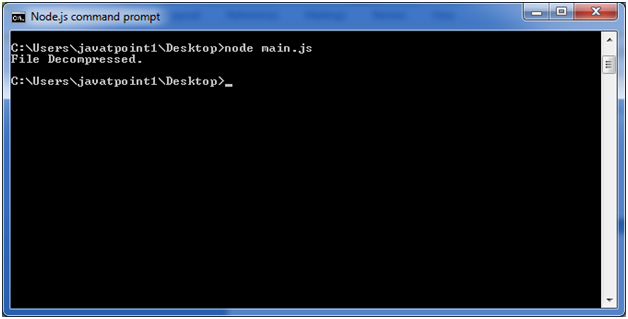
**To Decompress the same file:** put the following code in the js file "main.js"

*File: main.js*

1. var fs = require("fs");
2. var zlib = require('zlib');
3. // Decompress the file input.txt.gz to input.txt
4. fs.createReadStream('input.txt.gz')
5. .pipe(zlib.createGunzip())
6. .pipe(fs.createWriteStream('input.txt'));
7. console.log("File Decompressed.");

Open the Node.js command prompt and run main.js

1. node main.js



Node.js File System (FS)

In Node.js, file I/O is provided by simple wrappers around standard POSIX functions. Node File System (fs) module can be imported using following syntax:

**Syntax:**

1. var fs = require("fs")

Node.js FS Reading File

Every method in fs module has synchronous and asynchronous forms.

Asynchronous methods take a last parameter as completion function callback. Asynchronous method is preferred over synchronous method because it never blocks the program execution where as the synchronous method blocks.

Node.js Open a file

**Syntax:**

Following is the syntax of the method to open a file in asynchronous mode:

1. fs.open(path, flags[, mode], callback)

**Parameter explanation:**

Following is the description of parameters used in the above syntax:

**path:** This is a string having file name including path.

**flags:** Flag specifies the behavior of the file to be opened. All possible values have been mentioned below.

**mode:** This sets the file mode (permission and sticky bits), but only if the file was created. It defaults to 0666, readable and writeable.

**callback:**This is the callback function which gets two arguments (err, fd).

Node.js Flags for Read/Write

Following is a list of flags for read/write operation:

|  |  |
| --- | --- |
| **Flag** | **Description** |
| r | open file for reading. an exception occurs if the file does not exist. |
| r+ | open file for reading and writing. an exception occurs if the file does not exist. |
| rs | open file for reading in synchronous mode. |
| rs+ | open file for reading and writing, telling the os to open it synchronously. see notes for 'rs' about using this with caution. |
| w | open file for writing. the file is created (if it does not exist) or truncated (if it exists). |
| wx | like 'w' but fails if path exists. |
| w+ | open file for reading and writing. the file is created (if it does not exist) or truncated (if it exists). |
| wx+ | like 'w+' but fails if path exists. |
| a | open file for appending. the file is created if it does not exist. |
| ax | like 'a' but fails if path exists. |
| a+ | open file for reading and appending. the file is created if it does not exist. |
| ax+ | open file for reading and appending. the file is created if it does not exist. |

Node.js File Information Method

**Syntax:**

Following is syntax of the method to get file information.

1. fs.stat(path, callback)

**Parameter explanation:**

**Path:**This is string having file name including path.

**Callback:**This is the callback function which gets two arguments (err, stats) where stats is an object of fs.Stats type.

Node.js fs.Stats class Methods

|  |  |
| --- | --- |
| **Method** | **Description** |
| stats.isfile() | returns true if file type of a simple file. |
| stats.isdirectory() | returns true if file type of a directory. |
| stats.isblockdevice() | returns true if file type of a block device. |
| stats.ischaracterdevice() | returns true if file type of a character device. |
| stats.issymboliclink() | returns true if file type of a symbolic link. |
| stats.isfifo() | returns true if file type of a fifo. |
| stats.issocket() | returns true if file type of asocket. |

Node.js Path

The Node.js path module is used to handle and transform files paths. This module can be imported by using the following syntax:

**Syntax:**

1. var path =  require ("path")

Node.js Path Methods

Let's see the list of methods used in path module:

|  |  |  |
| --- | --- | --- |
| **Index** | **Method** | **Description** |
| 1. | path.normalize(p) | It is used to normalize a string path, taking care of '..' and '.' parts. |
| 2. | path.join([path1][, path2][, ...]) | It is used to join all arguments together and normalize the resulting path. |
| 3. | path.resolve([from ...], to) | It is used to resolve an absolute path. |
| 4. | path.isabsolute(path) | It determines whether path is an absolute path. an absolute path will always resolve to the same location, regardless of the working directory. |
| 5. | path.relative(from, to) | It is used to solve the relative path from "from" to "to". |
| 6. | path.dirname(p) | It return the directory name of a path. It is similar to the unix dirname command |
| 7. | path.basename(p[, ext]) | It returns the last portion of a path. It is similar to the Unix basename command. |
| 8. | path.extname(p) | It returns the extension of the path, from the last '.' to end of string in the last portion of the path. if there is no '.' in the last portion of the path or the first character of it is '.', then it returns an empty string. |
| 9. | path.parse(pathstring) | It returns an object from a path string. |
| 10. | path.format(pathobject) | It returns a path string from an object, the opposite of path.parse above. |

Node.js StringDecoder

The Node.js StringDecoder is used to decode buffer into string. It is similar to buffer.toString() but provides extra support to UTF.

You need to use require('string\_decoder') to use StringDecoder module.

1. **const** StringDecoder = require('string\_decoder').StringDecoder;

Node.js StringDecoder Methodsx

StringDecoder class has two methods only.

|  |  |
| --- | --- |
| **Method** | **Description** |
| decoder.write(buffer) | It is used to return the decoded string. |
| decoder.end() | It is used to return trailing bytes, if any left in the buffer. |