**Node JS**

Node JS is a very powerful JavaScript based framework /platform built on Google Chrome's JavaScript v8 Engine. It is used to develop I/O intensive web application like video streaming sites, single - page applications, and other web applications. Node.js is open source, completely free, and used by thousands of developers around the world.

**What is Node JS**

Node.js is a server side platform built on Google Chrome's JavaScript Engine (V8 engine). Node.js was developed by Ryan Dalh in 2009 and its latest version is v0.10.36. The definition of Node.js as supplied by its official documentations.

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

Node.js also provides a rich library of various JavaScript modules which simplifies the development of web application using Node.js to a great extent.

Node.js = Runtime Environment + JavaScript library.

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

The definitions given by its official documentation is a follows.

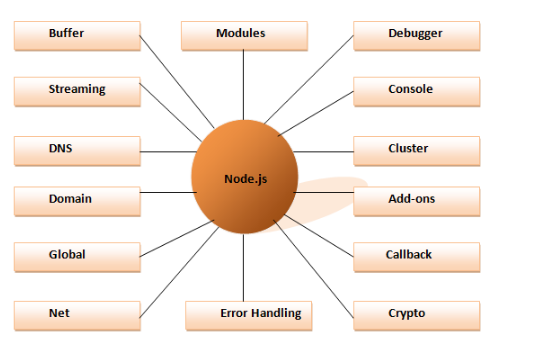
"*Node.js is a platform built on Chrome's JavaScript runtime for easily building fast and scalable network applications. Node.js uses a event-driven, non-blocking IO model that makes it light weighted and efficient, perfect for data intensive real-time application that run across distributed devices*".

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

Node.js = Runtime Environment + JavaScript Library

**Different parts of Node.js**

The following diagram specifies some important parts of Node.js



**Features of Node.js**

1. **Asynchronous and Event Driven** : All APIs of Node.js library are asynchronous that is, non-blocking. It essentially means 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 calls.

2. **Very Fast** : Being built on Google Chrome's V8 JavaScript engine Node.js library is very fast in code execution.

3. **Single threaded but highly scalable** : Node.js uses a single threaded model with event looping. Event mechanism helps to server to respond in a non-blocking way and makes the server highly scalable as opposed to traditional servers which create limited threads to handle requests. Node.js uses a single threaded program and the same program can provide service to a much larger number of request than traditional server like Apache HTTP Server.

4. **No Buffering** : Node.js applications never buffer any data. These applications simply output the data in chunks.

5. **License** : Node.js is released under the MIT license.

6. **Open Source** : Node.js has a open source community which has produced many excellent modules to add additional capabilities to Node.js applications.

**Where to use Node.js**

Following are the area where Node.js is proving itself as a perfect technology partner.

1. IO bound application.

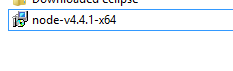
2. Data streaming application

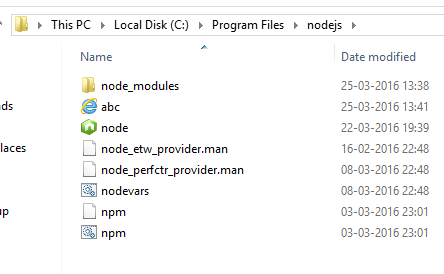
3. Data Intensive Real time Application (DIRT)

4. JSON APIs based applications.

5. Single page application.

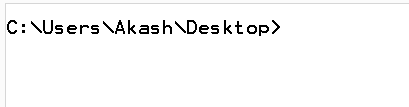
After download the Node.js





To get simple welcome message

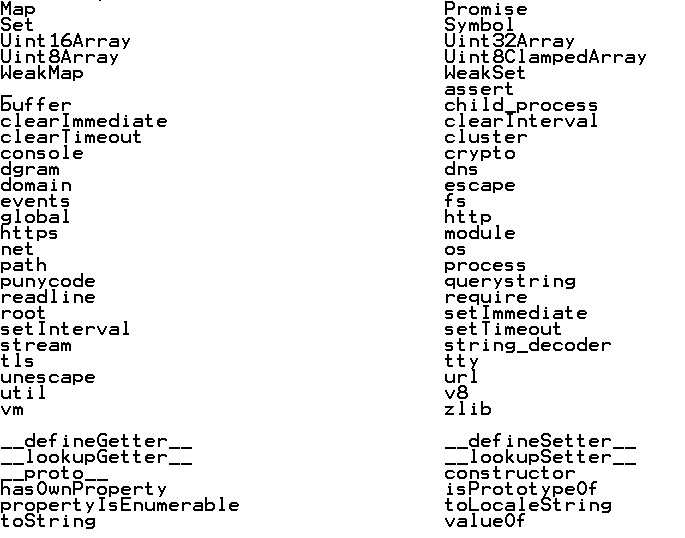
open the command prompt



Now type as **node**

then you will move to the node js environment

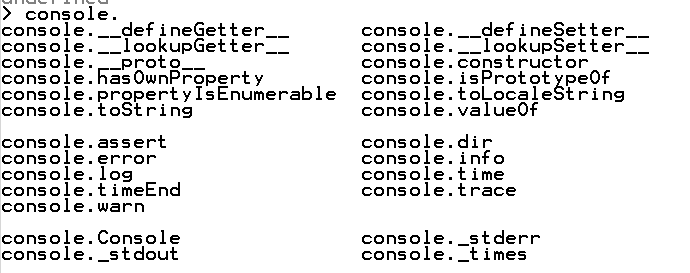
If you don't know any command then type as **tab** it will give the help of all commands.



Now to display message through node.js



If you don't know what console object contains the methods then first find the object name using **tab.** Then console. and again use tab to get the all methods belongs to console object.



Are else if you want to run the external file then

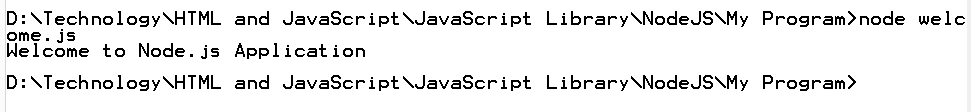
open the editor write the code and save the file with extension .js

**welcome.js**

console.log("Welcome to Node.js Application")

To run the file

node filename.js



**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 are used to write any node.js stream.

console.log()

console.error()

console.warn()

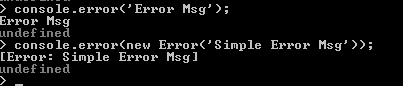
console.log()



To display the formatted output



console.error() method



console.warn() method



**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 REPL environment has a specific work.

Read : It reads user's input, parse the input into JavaScript data-structure and stored in memory.

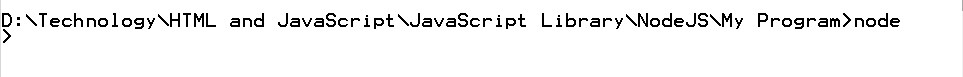
Eval : It takes and evaluates the data structure.

Print : It prints the result

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

**How to start REPL**

You can start REPL by simply running "node" on the command prompt



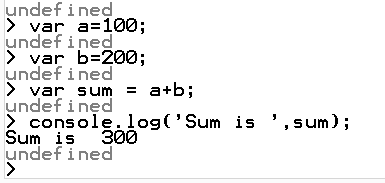
You can execute various mathematical operations on REPL Node.js command prompt.

**Node.js simple expression**

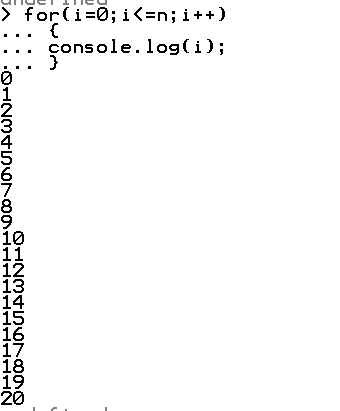


**Using variable**

Variable are used to store value and print later. If you don't user var keyword then value is stored in the variable and printed whereas is var keyword is used then value is stored not printed. You can print variable using console.log().

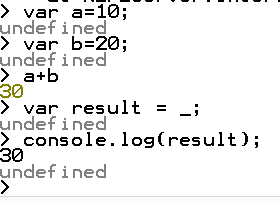


**Node.js multiline expression**



**Node.js underscore variable**

You can also use underscore \_ to get the result



**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. |

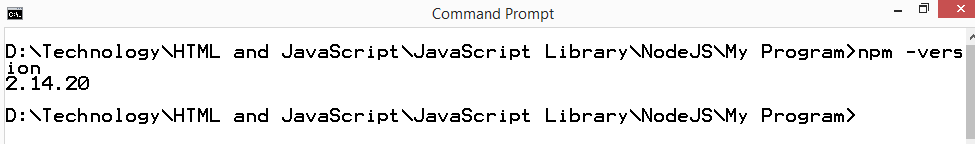
**Node.js package manager**

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 bundles with Node.js installable in versions after that v0.6.3. You can check the version by the opening Node.js command prompt and typing the command as



**installing Modules using npm**

Following is the syntax to install any Node.js module

npm install <moduleName>

A node.js application consists of following three important parts.

1 import required module : We use require directive to load a Node.js module.

2. create server : A server which will listen to client's request similar to Apache HTTP Server.

3. Read request and return response : Server created in earlier step will read HTTP request made by client which can be a browser to console and return the response.

**Import required modules :** We use require directive to load a Node.js module

**Create Server :** A server which will listen to client's request similar to Apache Http Server.

**Read request and return response :** Server created in earlier step will read HTTP request made by client which can be a browser or console and return the response.

**main.js**

var http = require("http");

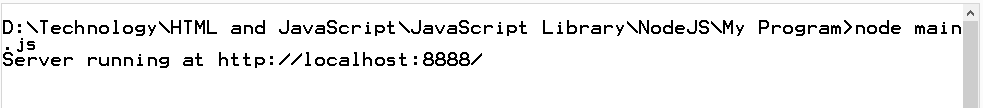
http.createServer(function(req,res){

res.writeHead(200,{"Content-Type":"text/plain"});

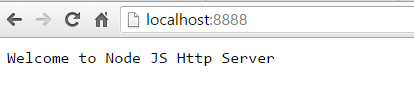
res.end('Welcome to Node JS Http Server');

}).listen(8888);

console.log('Server running at http://localhost:8888/')

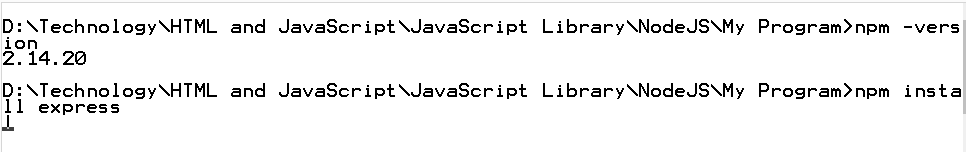


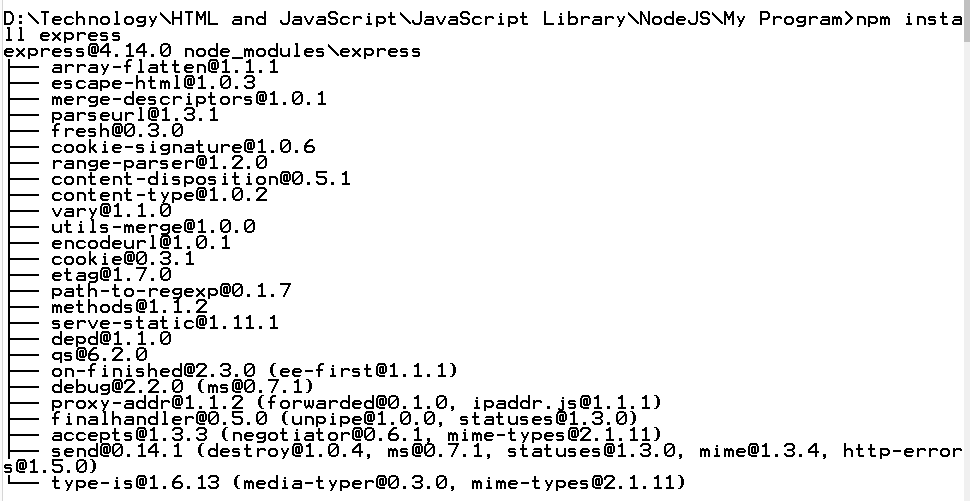
After running the program in browser



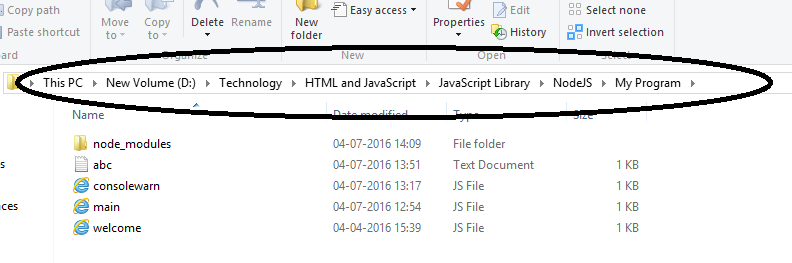
Let's install a famous Node.js web framework called express.

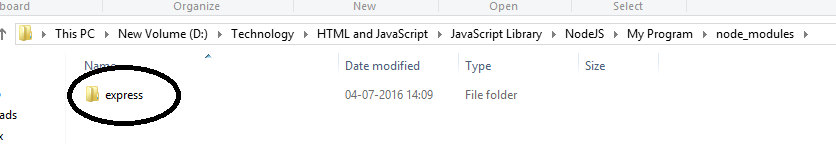
Open the Node.js command prompt and execute the following command.



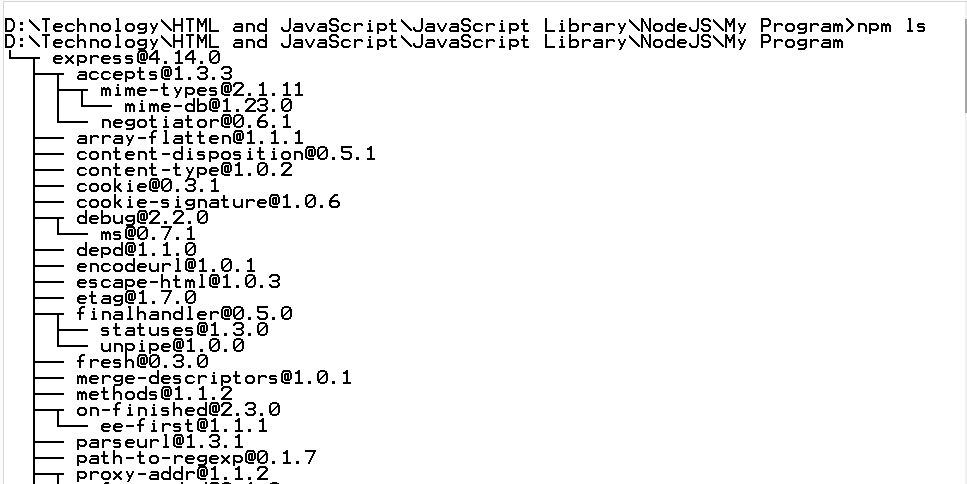


Here the module are installed



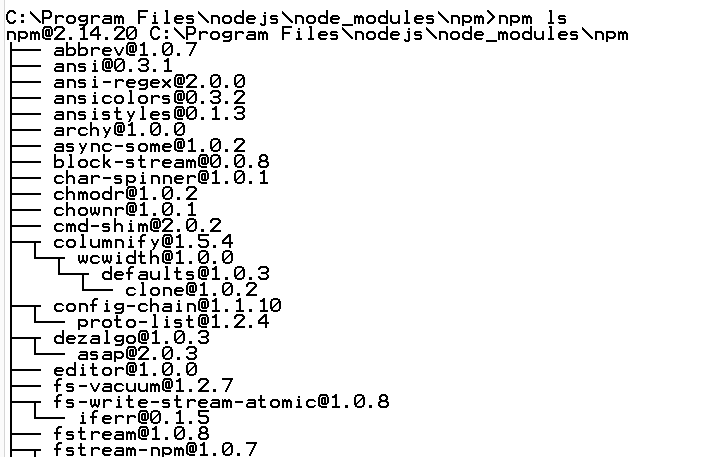


To see the list of module installed



This command are use to see the list of module installed locally

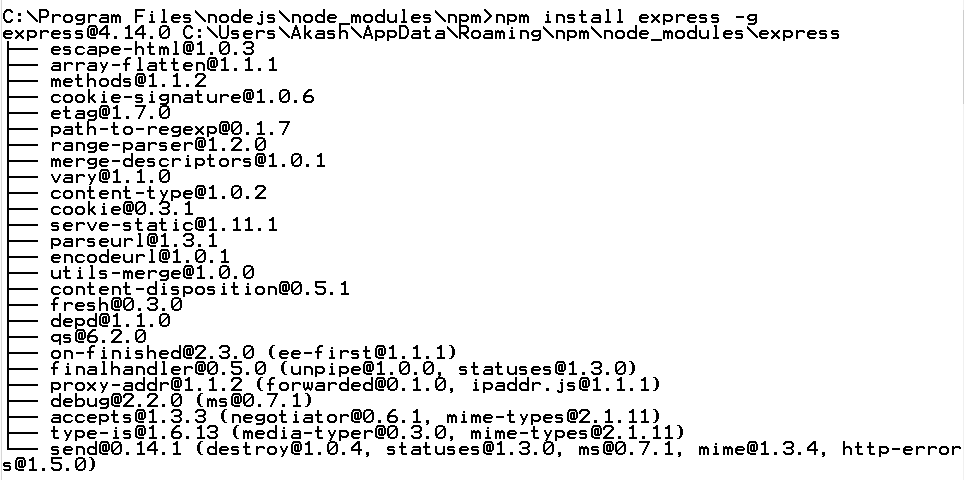
To see the list of module in Node.js software installed



Global 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

**npm install express -g**



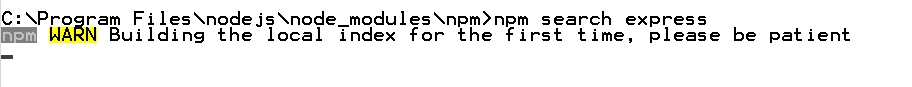
Here first line tells about the module version and its locations where it is getting installed.

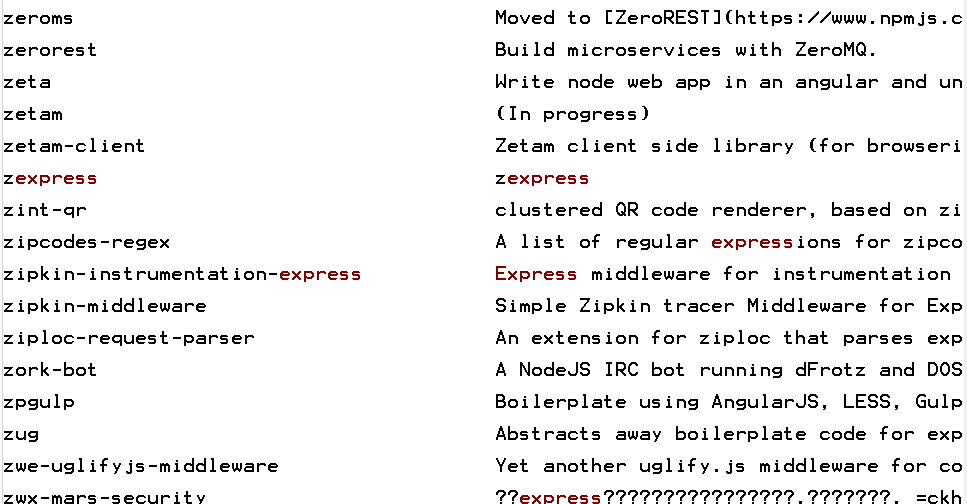
**Uninstalling a module**

npm uninstall express

**Searching a module**

"npm search express"





**Node.js with Passing the value through Get method**

**check.js**

var express = require('express');

var app = express();

app.get('/process\_post',function (req, res) {

var name = req.query["first\_name"];

var pass = req.query["last\_name"];

if(name==="Akash" && pass==="Kale"){

res.write("Successfully Login");

}else {

res.write("Failure try once again");

}

res.end("");

});

var server = app.listen(8989, function () {

var host = server.address().address;

var port = server.address().port;

console.log("Example app listening at http://%s:%s", host, port);

});

**Login.html**

<!DOCTYPE html>

<html>

<head>

<meta charset="ISO-8859-1">

<title>Insert title here</title>

</head>

<body>

<form action="http://localhost:8989/process\_post" method="get">

Name:<input type="text" name="first\_name"><br>

Password:<input type="password" name="last\_name"><br>

<input type="submit" value="submit">

</form>

</body>

</html>

**After running this program**

****

**Node.js with Passing the value through Post method**

var express = require('express');

var app = express();

var bodyParser = require('body-parser');

app.use(bodyParser.json()); // support json encoded bodies

app.use(bodyParser.urlencoded({ extended: true }));

app.post('/process\_post',function (req, res) {

var name = req.body.first\_name;

var pass = req.body.last\_name;

if(name==="Akash" && pass==="Kale"){

res.write("Successfully Login");

}else {

res.write("Failure try once again");

}

res.end("");

});

var server = app.listen(8989, function () {

var host = server.address().address;

var port = server.address().port;

console.log("Example app listening at http://%s:%s", host, port);

});

**Login.html**

<!DOCTYPE html>

<html>

<head>

<meta charset="ISO-8859-1">

<title>Insert title here</title>

</head>

<body>

<form action="http://localhost:8989/process\_post" method="post">

Name:<input type="text" name="first\_name"><br>

Password:<input type="password" name="last\_name"><br>

<input type="submit" value="submit">

</form>

</body>

</html>

After running this program



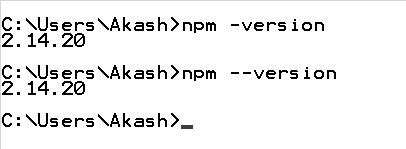
**Node.js Command Line Options**

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

|  |  |
| --- | --- |
| Option | Description |
| -version, --version | It is used to print node's version |
| -h, --help | It is used to print node command line options |
| -e, -eval | It evaluates the following arguments as JavaScript. The modules which are predefined in the REPL can also be used in script |
| -p, --print | It is identical to -e but prints the result |
| -i, --interactive | It opens the REPL even if stdin does not appear to be a terminal |
| -r, --required module | It is used to preload the specified module at startup. It follows require module resolution rules, Module may be either a path to a file, or node module name. |
| --no - deprecation | Silence deprecation warnings |
| --trace - deprecation | It is used to print stack traces for deprecation |
| --throw -deprecation | It silence all process warnings (including deprecations) |
| --trace -warning | It prints stack traces for process warnings(including depreactions) |
| --trace -sync-io | It prints a stack trace whenever synchronous i/o it detected after the first turn of the even loop. |
| --zero-fill-buffers | Automatically zero-fills all newly allocated buffer and slow buffers instances |
| --track - heap objects | It tracks heap object allocations for heap snapshots |
| --prof-process | It process V8 profiler output generated using the v8 option --prof |
| --V8-options | It prints V8 command line options |
| --tls-chiper-list = list | It specifies an alternative default tls cipher list. ( requires node.js to be built with crypto support\_. |
| --enables-fips | It forces fips compliant on startup. |
| --icu-data-dir=file | It specifies data load path. |

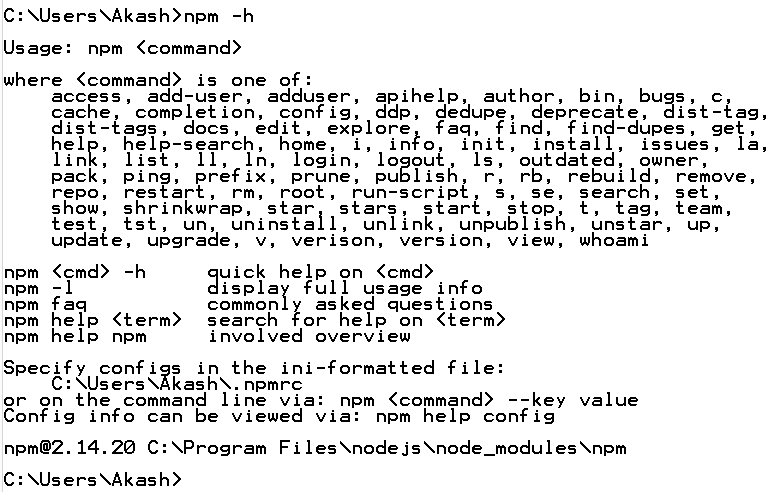
**npm --version**

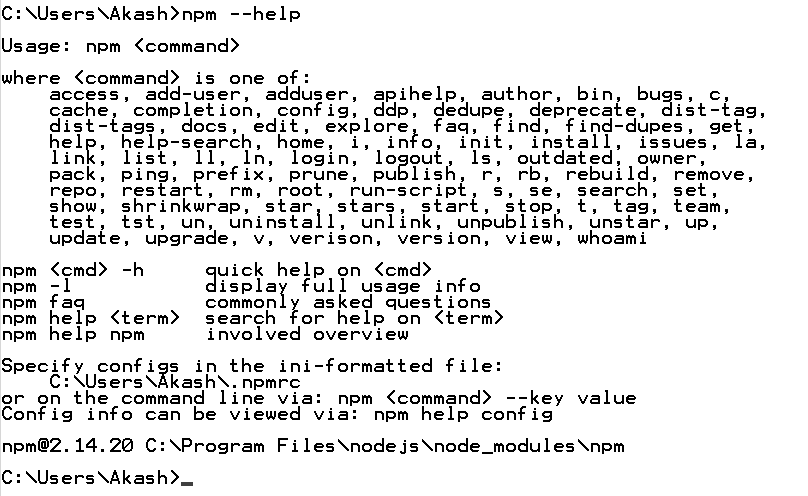
**npm -version**

****

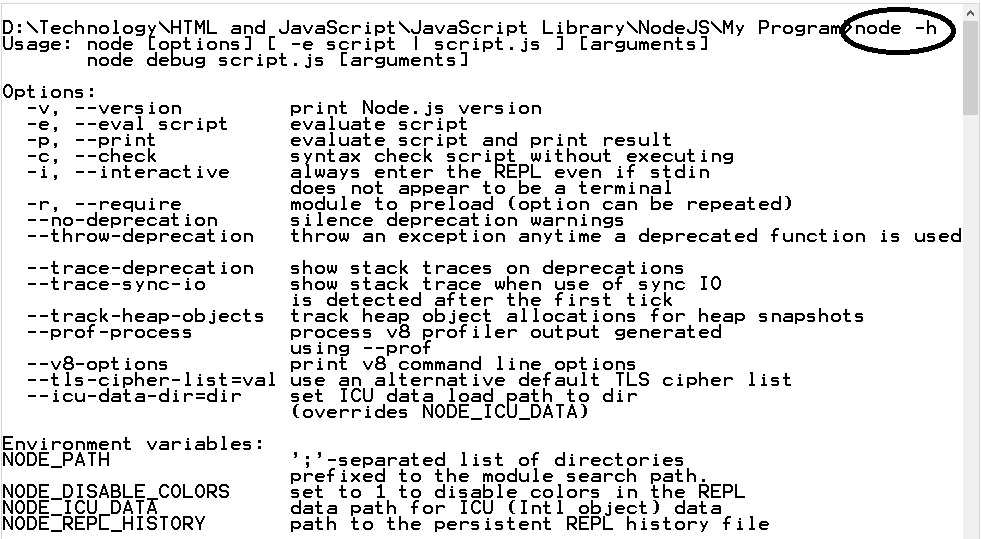
**npm -h**

**npm --help**



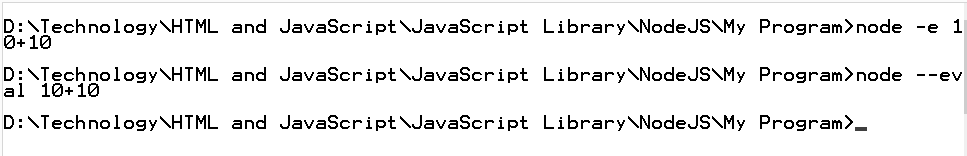


As we as

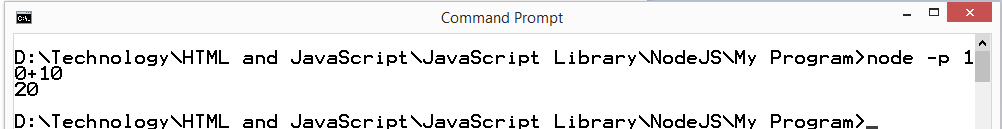


**node -e**

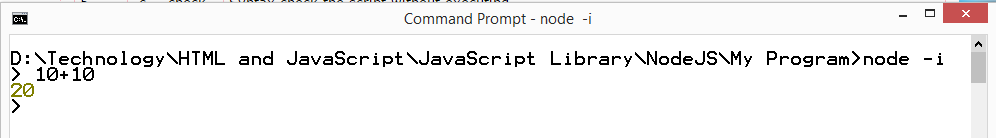
**node --eval**



**node - p**



**node - i**



**Node.js Global Objects**

Node.js global object 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 objects 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.

1. \_\_dirname( two under score)

2. \_\_filename

3. console

4. process

5. Buffer

6. setImmediate(callback[,arg][,])

7. setInterval(callback,delay[],[])

8.setTimeout(callback,delay[],[])

9.clearImmedate(immedaiteObject)

10. clearInterval(intervalObject)

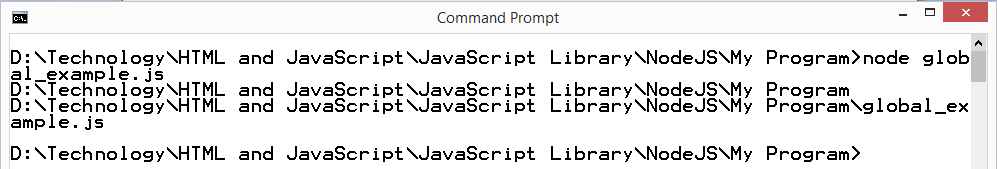
11.clearTimeout(timeoutObject)

**\_\_dirname and \_\_filename global objects**

**global\_example.js**

console.log(\_\_dirname)

console.log(\_\_filename)



**Node.js Buffers**

Node.js provides Buffer class to store raw data similar to an array of integer but corresponding 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 is necessary to handle octet streams.

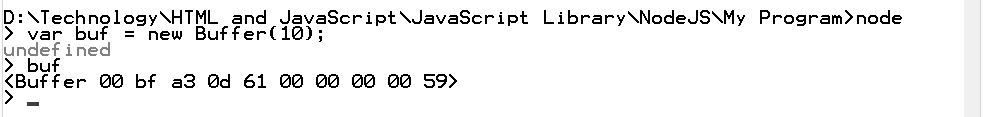
Buffer class is global class. It can be accessed in application without importing buffer module.

**Node.js creating Buffers**

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

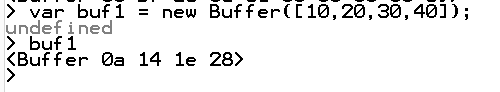
1. Create an uninitiated buffer : Following in the syntax of creating an uninitiated buffer of 10 octects.

var buf = new Buffer(10);



2. Create a buffer from array : Following is the syntax to create a Buffer from a given array.

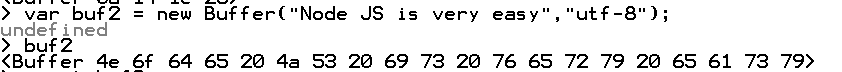
var buf = new Buffer([10,20,30,40,50]);



Creating a buffer from string :

Following is the syntax to create a Buffer from a given string and optionally encoding type.

var buf2 = new Buffer("Simple Easy Learing","utf-8")



**Node.js writing to buffers**

Following is the method to write into a Node buffer

syntax

buf.write(string,[,offest],[,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. Default to buffer.length.

encoding : Encoding to use. "utf-8" 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.

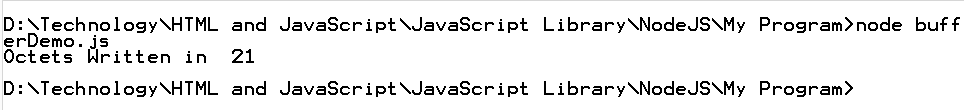
**bufferDemo.js**

var buf = new Buffer(256);

var len = buf.write("NodeJS is very simple");

console.log("Octets Written in "+len)

Run the file.



**Node.js Reading from buffers**

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

syntax

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

**Parameter explanation**

encoding : It specifies encoding to use. UTF-8 is the default encoding.

start : It specifies beginning index to start reading, default to 0.

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

**Return value reading from buffers**

**ReadingBuffer.js**

buf = new Buffer(26);

for (var i = 0 ; i < 26 ; i++) {

buf[i] = i + 97;

}

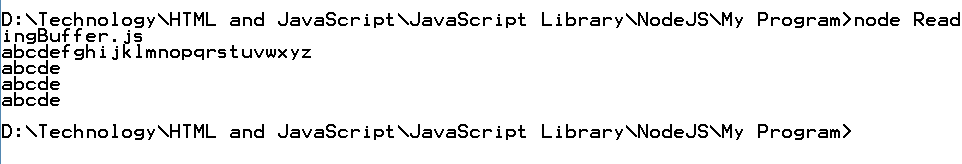
console.log( buf.toString('ascii')); // outputs: abcdefghijklmnopqrstuvwxyz

console.log( buf.toString('ascii',0,5)); // outputs: abcde

console.log( buf.toString('utf8',0,5)); // outputs: abcde

console.log( buf.toString(undefined,0,5)); // encoding defaults to 'utf8', outputs abcde

After running this program



**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**

1. setImmediate(): It is used to execute setImmediate

2. setInterval() : It is used to define a time interval

3. setTimeout() : It is used to execute a one - time callback after delay milliseconds.

**Clear timer functions**

1. clearImmediate(immediateObject) : It is used to stop an immediateObject, as created by setImmediate.

2. clearInterval(intervalObject): It is used to stop an intervalObject, as created by setInterval.

3. clearTimeout(timeoutObject): It prevents a timeoutObject, as created by setTimeout.

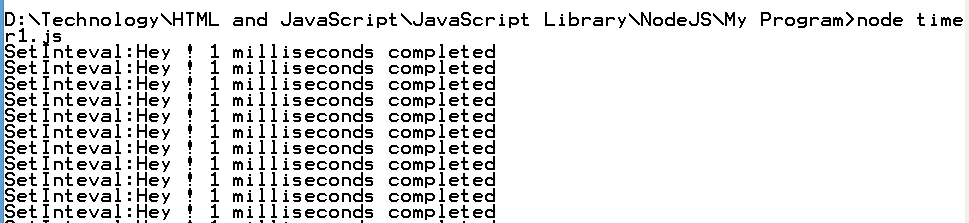
**timer1.js (setInteval() function)**

setInterval(function(){

console.log("SetInteval:Hey ! 1 milliseconds completed");

},1000)

After running this program



**timer2.js (setInteval() function)**

var i=0;

console.log(i);

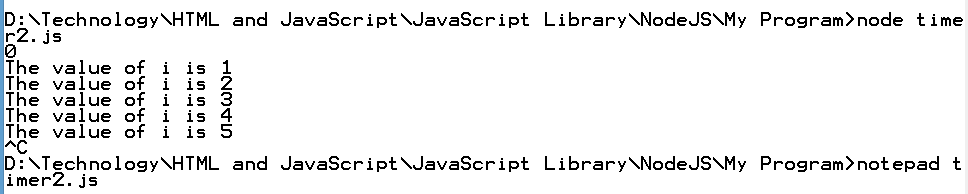
setInterval(function(){

i++;

console.log("The value of i is "+i);

},1000)

After running this program



**timer3.js (setTimeout() function)**

var i=0;

console.log(i);

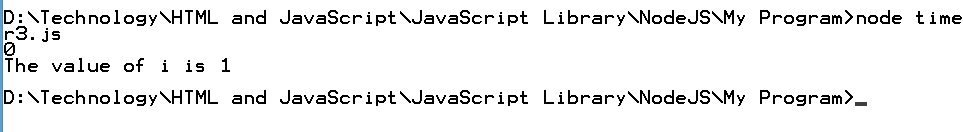
setTimeout(function(){

i++;

console.log("The value of i is "+i);

},1000);

After running this program



This example shows time out after every 1000 milliseconds without setting a time interval. This example uses the recursion property of a function

**timer4.js**

var recursive = function(){

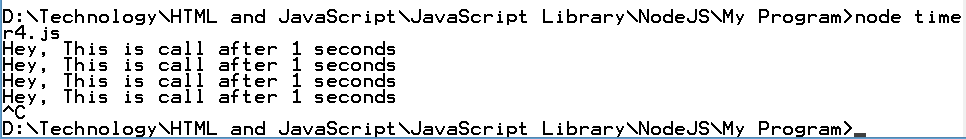
console.log("Hey, This is call after 1 seconds")

setTimeout(recursive,1000)

}

recursive();

After running this program



**Node.js setInterval(), setTimeout() and clearTimeout()**

var i=0;

function welcome() {

if(i<=5){

console.log("Welcome to Node JS "+i);

}

i++;

}

var id1 = setTimeout(welcome,1000);

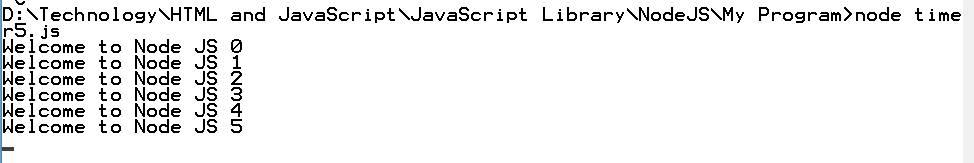
var id2 = setInterval(welcome,1000);

if(i>=5){

clearInterval(id2);

}

After running this program



**Node.js Errors**

The Node.js applications generally face four type of errors

1. Standard JavaScript errors

a. <EvalError>, <SyntaxError>, <RangeError>, <ReferenceError>, <TypeError>, <URIError> etc

2. System errors

3. User-specified errors

4. Assertion errors

**ReferenceError Example**

An EvalError indicates an error in the eval() function

Newer versions of JavaScript or Node js doesn't throw any EvalError. Use SyntaxError instead.

**RangeError**

A RangeError is thrown if you use a number that is outside the range of legal values.

**RangeError.js**

var num = 1;

try {

num.toPrecision(500);

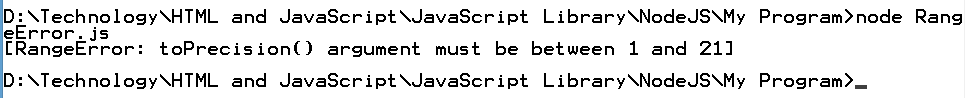
}

catch(err) {

console.log(err);

}

After running this program



**ReferenceError.js**

try{

const a = 1;

const b = a+c;

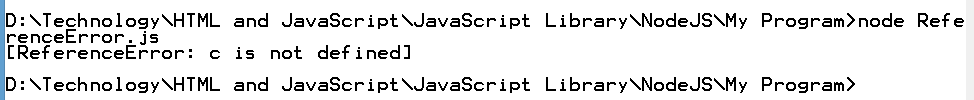
console.log("No Exception");

}catch(err) {

console.log(err);

}

After running this program



**Syntax Error**

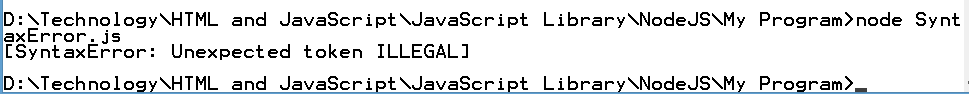
try{

eval("alert('Hi)")

}catch(err) {

console.log(err)

}



**Type Error**

A TypeError is thrown if you use a value that is outside the range of expected types.

**TypeError.js**

var a = 10;

try {

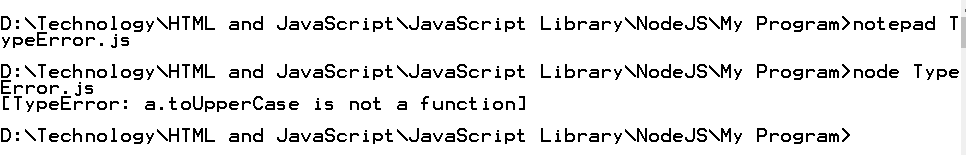
a.toUpperCase();

}catch(err) {

console.log(err);

}

After running this program



**try catch and finally block**

TryCatchFinally.js

var a = 10;

try {

//a.toUpperCase();

console.log("No Exception");

}catch(err) {

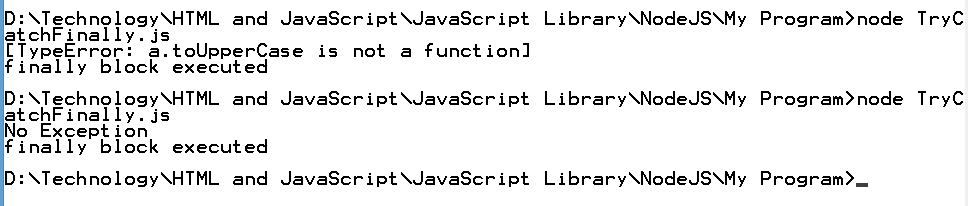
console.log(err);

}finally {

console.log("finally block executed");

}

After running this program



**throw exception**

var a =10;

var b = 20;

try {

if(b>a) {

//throw new "Error Generated"

//throw new EvalError();

throw new SyntaxError("My Syntax Generated");

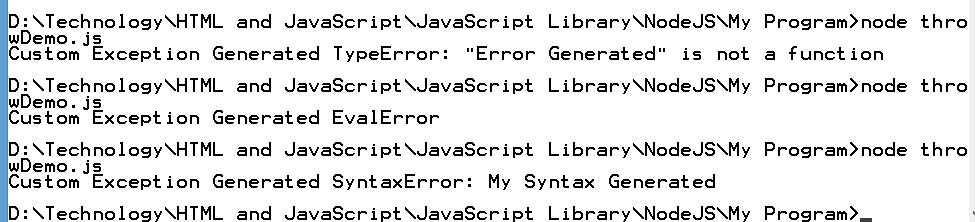
}

}catch(err) {

console.log("Custom Exception Generated "+err)

}

After running this program



**Node.js DNS (Domain Name Server)**

The Node.js DNS module contains methods to get information of given hostname. Let ' s see the list of commonly use 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.resolveSrv(hostname,callback)

dns.resolvePtr(hostname,callback)

dns.resolveTxt(hostname,callback)

dns.reverse(ip,callback)

**DNS Modules**

This modules provides functions to perform operations on domain names. These function are only available after a require("dns"); statement.

**Node.js dns.lookup() function**

This functions resolves a given domain name into an IPV4 or IPV6 address. This functions uses the functions of underlying operating system which interact with a network. This functions does not directly interact with a network. This function is the only function in this module which does not interact with a network directly.

The node.js dns.lookup() accepts three arguments.

1. The first argument is the domain name or host name which needs to be resolved.

2. The optional second argument is options. This second argument can take integer values like 4 and 6 which denotes the IP version numbers. This second argument can also be an options object which holds two optional properties. The first property of options object is family, it takes 4 or 6 as its value and denotes the IP version number. The second property of options object is hints.

3. The third argument for the node.js dns.lookup() function is the callback. This callback receives three arguments. The first argument is the error if the functions encounters any errors. The second argument is the resolved address. The third argument is the family which denotes whether the resolved address is IPV4 or IPV6.

dns.lookup() function return an IPV4 or IPV6 address.

**dnslookup.js**

var dns = require("dns");

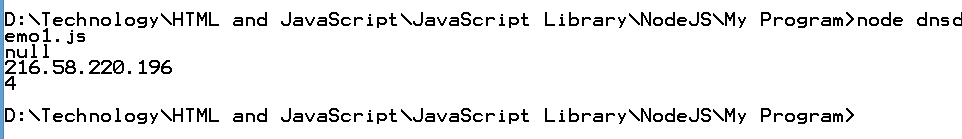
dns.lookup("www.facebook.com",function(err,address,family){

console.log("Address "+address);

console.log("Family"+family);

});

After running this program



**Node.js dns.resolve() function**

This function returns the DNS records of a given host name or domain name by interacting with the network directly. This function accepts three argument.

The first argument is the hostname or domain name which needs to resolved.

The optional second argument is the resolve record type which specifies the record which is required.

The third argument is the callback. This callback receives two arguments. The first argument to callback is the error and the second argument to callback is the resolved records.

**dnsresolve.js**

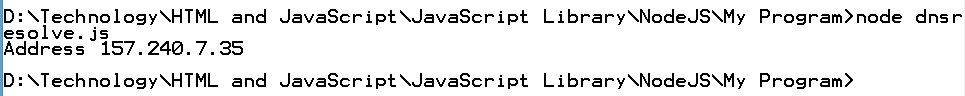
var dns = require("dns");

dns.resolve("www.facebook.com",function(err,address,family){

console.log("Address "+address);

});

After running this program



**Node.js dns.reverse() function**

This function resolves an IP address into a host name or domain. This function accepts two arguments. The first argument is the IP address that needs to be resolved. The second argument is the callback. This callback receives two arguments. The first argument to callback is the error if the function encounter an error. The second argument to the callback is an array of resolved host names.

**dnsreverse.js**

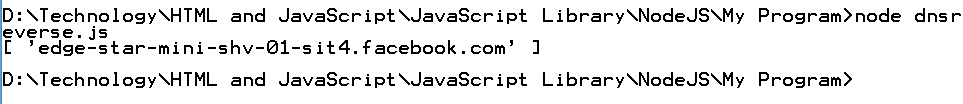
var dns = require("dns");

dns.reverse("31.13.78.35", function(err, hostnames) {

console.log(hostnames);

});

After running this program



**Node.js TCP Network (net) module**

The net or network modules contains functions to create servers and clients which can communicate over a network using TCP.

**Node.js Sockets**

Sockets are endpoints of communication in a network. In a network, data is sent between one socket and another. A server has its socket and a client also has its socket. All data that a client or server sends is sent through a socket and all data that a client or server receives is received through a socket.

In node.js socket are duplex streams. Data can be read from a socket as a stream and data can also be written to a socket as a stream. A stream written to a socket is sent over a network to another socket where the stream will be readable.

All the sockets created by node.js **net** module are TCP sockets. All sockets created by node.js **dgram** module are UDP sockets.

**Node.js TCP Server**

A server is a network entity which accepts connection and communicates with connection entities. A server accepts connections from a specified port on a host, domain or an IP address.

A node.js Server is an object or net.Server class. This server object should be set to listen to connections on a port and host combination. A node.js server creates an instance or a socket for each new connection. Any stream written to an instance of a socket is sent over its corresponding connection.

**Node.js TCP client**

A client is a network entity which makes a connections to a server and communicates with it. A client connects to a server residing at a port on a host, domain or an IP Address.

A node.js Client is a socket object of net.socket class. This client socket object connects to a port and host combination. A client has a different socket for each connection that it makes. Any stream written to an instance of a client socket is sent over its corresponding connection to server.

**Node.js net.createServer() function**

This function creates a new TCP server. This function accepts two optional arguments. This function returns a new server object.

The first argument is the options object which contains two optional properties. The first property of the options object is allowHalfOpen. It takes a boolean value which decides whether the server socket can stay open even if the client socket which it is communicating with is closed. The second property of the option object is pauseOnConnect. It takes a boolean value which decides whether to pause the socket stream when a new connection is received.

The second argument is a listener function of connection event. This function receives a socket as its argument. A new socket instance is created for each new connection.

**Node.js net module - connection event**

The server emits a connection event every time a new connection is received.

**Node.js net module - server.listener() function**

This function allows a server to accept new connection. This is a member function of the server object. This function is an overloaded function and most widely used format of this function accepts two arguments.

The first argument is port and accepts a numerical value which is the port number on which new connections are to be accepted by the server.

1 port - Optional

2 host - Optional

3 backlog - Optional

4 path - Optional

5 exclusive Boolean - Optional

Port property accepts a numerical value which is the port number on which new connections are to be accepted by the server.

The second argument is host and accepts a string value which is the host name or domain name on which new connections are to be accepted by the server. If this property is skipped, it defaults to localhost.

The third argument is backlog and it accepts a number value which is the maximum number of connection that can be queued at any given time.

The fourth argument to the listen() function is a listener to the "listening" event.

**Node.js new module - listening event**

The "listening" event is emitted upon successful execution of this listen() function and indicates that server is ready to accept connections from client.

**netServer.js**

var net = require("net");

var server = net.createServer();

server.listen(8787,"localhost",2);

server.on("listening",function() {

console.log("server accepting connections");

});

server.on("connection",function(server\_socket){

console.log("client connected");

server\_socket.write("Message from server");

});

**netClient.js**

var net = require("net");

var client\_socket = net.connect({port:8787});

client\_socket.on("connect",function() {

console.log("Connected to server !");

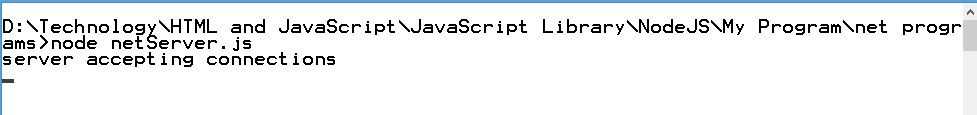
});

client\_socket.on("data",function(data){

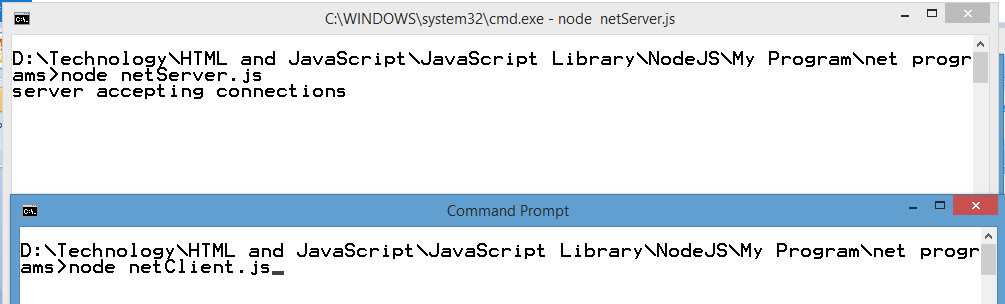
console.log(data.toString());

});

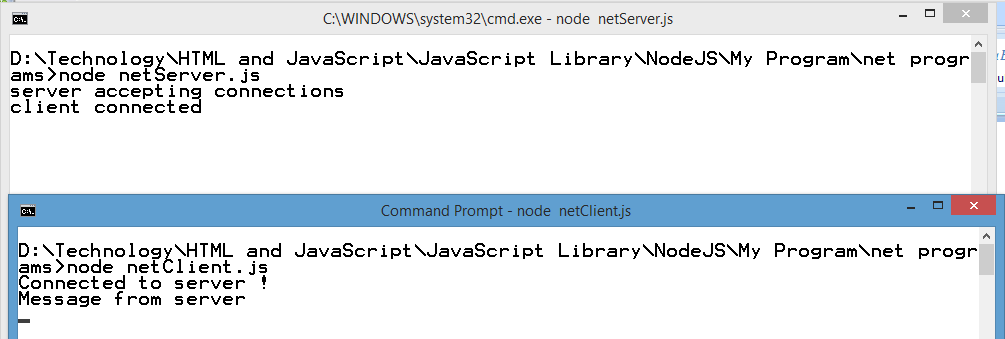
So first run the Server program



Then run the client console



After hit the client console



**Another Chatting Program**

**netServer1.js**

var net = require("net");

var server = net.createServer();

server.on("connection", function(server\_socket) {

console.log("client connected");

server\_socket.write("Hello client!");

server\_socket.pipe(server\_socket);

});

server.listen(8888,"localhost",2);

server.on("listening", function() {

console.log("server accepting connections");

});

**netClient1.js**

var net = require("net");

var client\_socket = net.connect({port:8888});

client\_socket.on("connect", function() {

console.log("connected to server!");

});

process.stdin.pipe(client\_socket);

client\_socket.on("data", function(data) {

console.log(data.toString());

});

**Node.js net module - server.close() function**

This function stops a server from accepting any new connections. The existing connections will function normally. The server will close after all the existing connections end and emits a close event.

**Node.js close TCP server - serverClose.js**

var net = require("net");  
var server = net.createServer();  
server.on("connection", function(server\_socket) {  
console.log("client connected");  
});  
server.listen(8888,"localhost",2);  
server.on("listening", function() {  
console.log("server accepting connections");  
server.close();  
});  
server.on("close", function(){  
console.log("server is closed and will not accept any more connections");  
});

**Node.js net module - server.address() function**

This function returns the address on which a server is accepting new connections.

**serverAddress.js**

var net = require("net");  
var server = net.createServer();  
server.on("connection", function(server\_socket) {  
console.log("client connected");  
});  
server.listen(80,"localhost",2);  
server.on("listening", function() {  
console.log("server accepting connections");  
console.dir(server.address());  
server.close();  
});  
server.on("close", function(){  
console.log("server is closed and will not accept any more connections");  
});

**Node.js net module - server.unref() function**

This function closes a server if that server is the only thing which is stopping a node process from existing.

**Node.js net module - server.ref() function**

This function removes an unref() on a server.

**Node.js net module - server.maxConnection property**

This is property of a server object. The maximum number of connections to a server at any given time can be limited by setting this property.

**Node.js net module - server.getConnection() function**

This function obtain the number of connection to a server. This function requires a callback as its arguments.

**Node.js Crypto**

The Node.js crypto module supports cryptography. It provides cryptographic functionality that includes a set of wrappers for open SSL's has HMAC, Cipher, Decipher, sign and verify function

**What is Hash**

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

**What is HMAC**

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

**CryptoHmac.js**

var crypto = require("crypto");

var secret = "abcdefg";

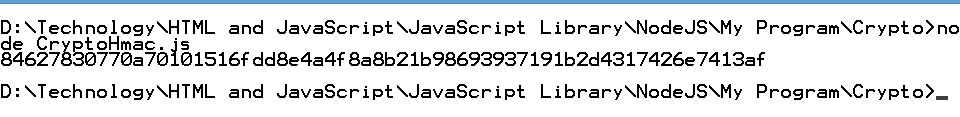
var hash = crypto.createHmac("sha256", secret)

.update("Welcome to JavaTpoint")

.digest("hex");

console.log(hash);

After running this program



**EncryptedAndDecrypted.js** Program

var crypto = require("crypto");

//Below code to convert encrypted data

var cipher = crypto.createCipher("aes192", "a password");

var encrypted = cipher.update("Hello, Akash Here!", "utf8", "hex");

encrypted += cipher.final("hex");

console.log(encrypted);

console.log("Data Encrypted Successfully");

console.log();

//Below code to convert decrypted data

var decipher = crypto.createDecipher("aes192", "a password");

var encrypted\_data = encrypted;

//var encrypted\_data = "abc123";

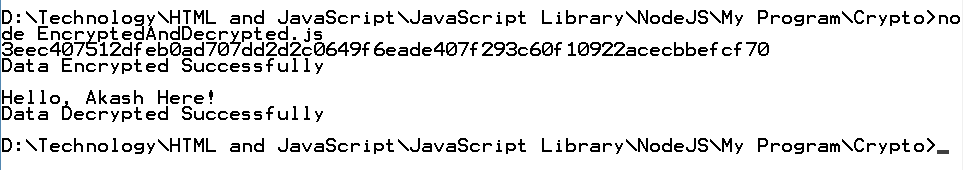
var decrypted = decipher.update(encrypted\_data, "hex", "utf8");

decrypted += decipher.final('utf8');

console.log(decrypted);

console.log("Data Decrypted Successfully");

After running this program



**Node.js TLS / SSL**

This topic we will later on

**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

**node debug script.js**

//console.log("Welcome to this file");

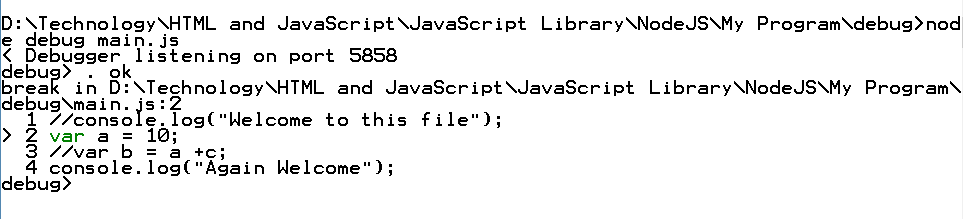
var a = 10;

//var b = a +c;

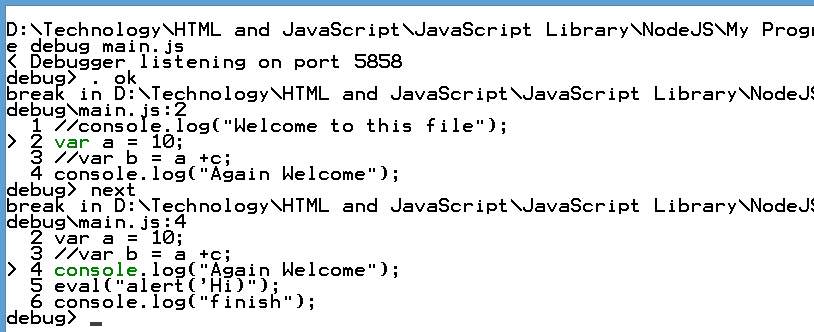
console.log("Again Welcome");

eval("alert('Hi)");

console.log("finish");



Now write next to go to next step.



**continue** – cont, c  
**step** – next, n  
**step in** – step, s  
**step out** – out, o

**Node.js Process**

Node.js provides the facility to get process information such as process id, architecture, platform, version, release, uptime, cpu usage etc. it can also be used to kill the process, set uid, set group, 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 | Return process architecture, arm, x64 |
|  |  |

**processDemo.js**

console.log('Process Architecture ',process.arch)

console.log('Process Id is ',process.pid)

console.log('Process Environment ',process.pid)

console.log('process platform ',process.platform)

console.log('process version',process.version)

console.log('process release',process.release)

**Node JS with Database - My SQL**