# Node.js Getting Started

The following is a list of sections in this document. Using Microsoft Word, you can use these as hyperlinks to navigate to any particular section. But using Apache Open Office, these hyperlinks do not work; instead, they merely serve as a table of contents. You can navigate to the start of any section via bookmarks; type F5 to bring up the Navigator; then double-click Bookmark1 for 1st section header, Bookmark 2, for 2nd section header, etc.

|  |  |
| --- | --- |
| [Course Introduction](#_Course_Introduction) | [What is Node?](#_What_is_Node?) |
| [Why Node?](#_Why_Node?) | [Some Analogies About Node](#_Some_Analogies_About) |
| [What You Get When You Install Node](#_What_You_Get) | [Example Files](#_Example_Files) |
|  |  |
|  |  |
|  |  |
|  |  |

## Course Introduction

Much of this clip deals with Node.js under Windows versus Node.js under Linux. The following are some statements made by the Pluralsight instructor on this subject.

At 2:31 the instructor brings up the subject of Windows versus Linux.

“Node itself is a bit different on Windows than it is on Linux and Mac.”

This confirms, at least, that Node does run on Windows.

At 2:47 the instructor is commenting about installing the Windows Subsystem for Linux.

“If you have a modern Windows machine, one option that might work a lot better for you is to install the Windows subsystem for Linux. This option will give you the best of both worlds. You’ll have your Windows operating system running Linux without needing to reboot, so you can work a Linux file system with your Windows editor, for example, which I think is great. I’ve tested this option, and I can confidently say that this will probably be the future of writing code for Node on Windows. Node is usually deployed on Linux machines in production, so by using a Linux environment on your Windows machine, you’ll be closer to the way your applications are running in production, and that’s always a good win.”

At least the instructor did not rule out the possibility of deploying a production Node application on a Windows machine.

The Pluralsight instructor strongly encouraged the student to install the **Windows Subsystem for Linux** if the Windows operating system is a recent vintage.

* Download the package using Google with the keywords “windows”, “subsystem”, “for”, “linux”. I found the website titled “Install WSL on Windows 10 | Microsoft Docs”.

***The next 2 steps – Windows Insider Program and preview Build of Windows 10 – may not have been necessary, because the approach that I tried (using the command “wsl --install”) did not work.***

* Navigate to a page titled Windows Installation Guide for Windows 10. This page strongly encourages the developer to join the Windows Insider Program. I did this.
* The next step is to Install a preview Build of Windows 10 (OS build 20262 or higher). I researched the web about how to perform this step.
  + Right-click the **Start** menu; choose **Run**. In the **Run** window, type **winver**. Windows displays a popup window that contains the version number. In my case the version number began with “19”, which convinced me that I needed a preview Build.
  + Click Start > Settings > Update & Security > Windows Update > Check for Updates (command button). Windows displays information that indicates that a preview build is available.
  + Close all open windows except the current window, and follow instructions to install the preview build. (It will be necessary to reboot the computer).
* Next I typed the command “wsl --install” at an administrative command prompt, and I repeatedly received an error indication that “wsl” is not recognized.

***Next I decided to follow the instructions for the “Manual Installation”.***

* Open a administrative command prompt, and type the command

dism.exe /online /enable-feature /featurename:Microsoft-Windows-Subsystem-Linux /all /norestart

* I decided to stick with WSL 1. Therefore, I shut down and restarted the computer.
* Navigate to the **Microsoft Store** to get WSL (<https://aka.ms/wslstore>). Then click the **Open Microsoft Store** command button. The browser shows a page titled “Run Linux on Windows”.
* Click **Ubuntu**. Then click the **Launch** command button. The response is a console window with the instruction “Installing, this may take a few minutes...” After a few minutes the console window displays “Installation successful” followed by instructions to supply a username and password.
* Follow the instructions, and keep a record of the username and password.
* That completes the installation of WSL. You will probably be asked to reboot the computer again.

## What is Node?

There is a virtual machine (commonly V8) running on your back-end server. Node instructs this virtual machine to interpret Java-script code, and Node relays the result to the user.

## Why Node?

* Node supplies a wrapper around V8 to facilitate running Java-script on your server.
* Node comes with some feature-rich built-in modules.
* All of the modules that come with Node offer asynchronous API’s; the user can employ these without worrying about threads.
* You can code add-ons in C++ to perform computations.
* Node has a powerful debugger and other utilities.
* Node comes with NPM.

## Some Analogies About Node

What exactly is a CALLBACK? A callback is a function, but not just any function. A callback is called at a “later” point in the time of a program. This is done using an asynchronous method. The Pluralsight instructor shows an example.

function cb ( data ) {

// do something with data

}

SomeAsyncMethod ( cb );

Whereas a call to a method can take arguments that the method uses as input or uses as a place to return results, the situation above is a little different. The argument cb above is a reference to a function. At some point when SomeAsyncMethod() is executing, it may choose to execute the function that was supplied in its argument list. cb() does what it was designed to do, and returns execution to SomeAsyncMethod().

What exactly is a PROMISE? ***(I have never seen anything like promises in other programming languages.)***  Imagine that a function call is made, and that the function is supposed to return something that you asked for. But in this case, you only receive what you asked for if the function is successful; if the function unsuccessful, you will get something else that denotes failure. ***(It seems to me that this is like calling a function that returns a variant. The variant can assume the type of what you asked for when the called function is successful; otherwise, the variant could be null.)***

## What You Get When You Install Node

This Pluralsight clip contains instructions for how to install Node. In case the Pluralsight student has already installed Node, the clip contains some tests to help the student gauge whether the installation is up-to-date, and whether the installation contains the options that will serve the student well while shadowing the course.

It turns out that I did have Node.js already installed.

The first time that I installed Node was when I was attending the Pluralsight course **Javascript: Getting Started**. The instructions for installing Node.js are in [Section = “Introduction to Java Script”, Clip = “Installing Development Software”]. My notes about this clip are at sncole\source\GitRepositories\WebAppCourseNotes\Javascript\_GettingStarted.docx, section = “New Tools”.

I installed Node.js more recently when I was attending the Pluralsight course **Node.js: The Big Picture**. The instructions for installing Node.js are in [Section = “Assembling a Development Toolset”, Clip = “Installing Node / Testing Applications”]. My notes about this clip are at sncole\source\GitRepositories\WebAppCourseNotes\Node\_js\_BigPicture.docx, section = “Testing Applications”.

The tests require the use of command-line interpreter. The command-prompt character shown by the Pluralsight course is “$”, which (at least to me) denotes “unix style”. I chose to use a Git Bash command interpreter. I found the instructions for obtaining the Git Bash command-line interpreter on the internet, and I have documented these instructions in sncole\source\GitRepositories\GitNotes\GitInstall.docx.

The tests are as follows.

Type

$ which node

You should see – as a response – a path ending with “/node”.

Type

$ which npm

You should see – as a response – a path ending with “/npm”.

Type

$ which npx

You should see – as a response – a path ending with “/npx”.

Type

$ node

You should see – as a response

Welcome to Node.js vnn.nn.nn

Type “.help” for more information.

>

“>” is a subordinate command prompt where you can type and execute Javascript.

***Clipboard operations on “>” commands are available by right-clicking the mouse on the “>” command; you will see a pop-up menu containing “Copy” and “Paste” (among others) as options.***

Navigate to <http://bit.ly/node-test> in a browser. The response will be a new web page titled “How modern is your Node?”. Copy – into the clipboard - the line of text that is immediately below “Modern JS test”, i.e. “(async (a = 1, ...b) => ({ ...b, a, [a]: `${a}` }))()”. Paste this line of text as a Javascript command (to the right of the “>” prompt). You should see the response “Promise { { '1': '1', a: 1 } }”.

If you get an error in the response, you should upgrade your version of Node.js; follow the Pluralsight instructions. Since I did not get an error, I bypassed the process of reinstalling Node.js.

Additional tests:

Copy – into the clipboard - the line of text that is immediately below “// New Promise APIs”, i.e. “util.promisify”. You should see the response “[Function: promisify] { custom: Symbol(nodejs.util.promisify.custom) }”

Copy – into the clipboard - the line of text that is 2 lines below “// New Promise APIs”, i.e. “require('fs').promises”. The response should be a comma-separated list of approximately 25 names.

Type “> .exit” to exit Node.js and return to the Git Bash command prompt. Type “$ exit” to exit Git Bash.

## Example Files

The Pluralsight instructor encourages the student to get sample code either from the Pluralsight **Exercise Files**, or from GitHub. I decided to try retrieving the code from GitHub.

Navigate - via a web browser - to <http://github.com/jscomplete/ngs> ; in response you should see the same web page as shown by Pluralsight. (“ngs” is an abbreviation of “node getting started”.) Clone this repository to a folder in the local computer. More specifically

* Start a Git Bash command-line interpreter on your local computer. You will see a “$” prompt.
* Apply “cd” (change directory) one or more times on Git Bash to navigate to a folder where you want to store the copied code. ***On my Windows-10 computer this folder is*** C:\users\sncole\source\GitRepositories\VSAndOtherPlatformProjects\NODE\_JS
* Return to the GigHub web page. Click the down-arrow in the green-background button. GitHub displays a popup dialog titled **Clone**.
* The popup dialog contains a URL followed by a small command button. Click the command button. GitHub displays “Copied!”.
* At the Gib Bash command-line interpreter, type “git clone “, then right-click the mouse. Next click **Paste** in the popup menu. You will see a copy of the URL after “git clone “. Finally type the **Enter** key. The “git clone” command responds by copying the repository to a subfolder titled “ngs”.
* I found it convenient to rename “ngs” to “ngs (Node Getting Started)”.

Navigate – via a file browser – to “ngs (Node Getting Started)”. You will see a list of files, some of which are preceded by numbers 1..6. These numbered folders contain subfolders, which – in turn – contain JavaScript (or related – e.g. “.json”) source files.