Vocabulary

Terminal -- The **terminal** is an interface that lets you access the command line on a computer.
Virtual Machine -- A virtual machine is an artificial computer inside a real computer. It doesn't exist physically but works using software. Several of these pretend computers can run on one actual computer. Each of them acts independently using different operating systems, almost like having different computers on the same machine.
Localhost -- a hostname that refers to the local machine currently making the request. Normally represented by the IPv4 address 127.0.0.1 or the IPv6 ::1.
Querystring -- A query string is a set of characters tacked onto the end of a URL.

The query string begins after the question mark (?) and can include one or more parameters. Each parameter is represented by a unique key-value pair or a set of two linked data items. An equals sign (=) separates each key and value. An ampersand (&) separates multiple parameters.

Static Resources -- resources that don't change and do not involve server-side code. This typically means images, CSS, and sometimes client-side Javascript.

Helper Function -- A helper function is a function that performs part of the computation of another function. Helper functions are used to make your programs easier to read by giving descriptive names to computations. They also let you reuse computations, just as with functions in general.

Getting Node

Node is easy to install. To install follow this three simple steps:

1. Go to Node home page.
2. Click the big green button that says INSTALL.
3. Follow installation instructions.

Windows & OS X: An installer will be downloaded that walks us through the process.
Linux: We should use a package manager. (Make sure to install the right version of Node.)

Using the Terminal

There is a lot of power and productivity in using a terminal (also referred to as a console or command prompt.)

For OS X or Linux there are a lot of useful shells. Popular ones are bash and zsh. For Windows users, because Windows suck ass we could install Git which includes “Git Bash” which provides a small subset of the normally available Unix commands. A recommendation for a better terminal for Windows is Console or ConEmu or the power shell.

Another option for the user is virtualization using virtual machines (VMs) which are powerful tool growing in popularity with the times. Linux VM or PuTTY is a great alternative to git bash which to be honest is just a bandage for Windows fuckupary.

Codio is another alternative. It will spin up a new Linux instance for every project you have and provide an IDE and command line, with Node already installed.

Editors

The editor is the programmer's primary tool. Finding a powerful editor and learning to use it will significantly increase your productivity.

Good editors are Vi (Vim), Emacs, Coda (OS X), Sublime Text, TextPad (Windows) Notepad++ (Windows), and Visual Studio.

npm

Npm is the ubiquitous package manager for Node packages. A package manager’s two primary responsibilities are installing packages and managing dependencies. npm is a fast, capable, and painless package manager.

npm is installed when you install Node. The primary command you’ll be using with npm is installed.

A Simple Web Server with Node

Usually web servers serve our computer static files so that a browser can view them over the network. Having a file with extension .html the web server simply knows where the file is on the computer, and serves it to the browser.

In Node the app that you write is the web server. Node provides the framework for us to build a web server. The best thing is that Node makes the business of writing this web server a simple affair and allows us to have more control of our application.

Hello World

In our favorite editor, let’s create a file called helloWorld.js.

Source: helloWorld.js

```
// Importing required packages
var http = require('http');

// Creates a server that can be listened at port 3000
http.createServer(function(req, res) {
      // sends a response header to the request
      res.writeHead(200, { 'Content-Type': 'text/plain' });
      // The res.end() function is used to end the response process with a
message.
      res.end('Hello World!');
}).listen(3000);

console.log('Server started on localhost:3000; press Ctrl-C to
terminate....');
```

In the same directory as helloWorld type in the terminal the following command

```
node helloWorld.js
```

Then open up a browser and navigate to *http://localhost:3000* to see our work. This example just transmits the message "Hello World!" in plaintext to the browser.

Event-Driven Programming

Event-Driven Programming is the core behind Node. To use Node effectively you have to understand what events are available to you and how to respond to them. A "click event" where the programmer has no idea of if or when the user will click a button, can be a good example.

In the prior example the event being handled is a HTTP request. The http.createServer method takes a function as an argument; that function will be invoked every time an HTTP request is made.

Routing

Routing is the mechanism for serving the client the content it has asked for. The client specifies the desired content in the URL in the path and querystring.

In the following example of "Hello World!" We are adding to our website a homepage, an about page, and a not found page.

***Source:** helloWorld.js*

```
// Importing required packages
var http = require('http');

// Creates a server that can be listened at port 3000
http.createServer(function(req, res) {
      // normalize url by removing querystring
      // trailing slash, and making it lowercase
      var path = req.url.replace(/\/?(?:\?.*)?$/,'').toLowerCase();
      switch(path) {
            // Empty querystring means "go to homepage"
            case '':
                  res.writeHead(200, { 'Content-Type': 'text/plain' });
                  res.end('Homepage');
                  Break;
            // An '/about' inside path will redirect us to the "About" page
            case '/about':
                  res.writeHead(200, { 'Content-Type': 'text/plain' });
                  res.end('About');
                  Break;
            // In the case the urlpath has something dumb like:
'asdalsjldasj'
            // it will return a status code of 404.
            default:
                  res.writeHead(404, { 'Content-Type': 'text/plain' });
                  res.end('Not Found');
                  break;
      }
}).listen(3000);

console.log('Server started on localhost:3000; press Ctrl-C to
terminate....');
```

Implementing these changes will allow us to browse the home page (*http://localhost:3000*), the About page (*http://localhost:3000/about*) any other querystring will return a 404 *http://localhost:3000/skldasjdlkasjlk*)

Serving Static Resources

In this section we will add some static resources like a real HTML and a logo image. Static resources with Node are doable only for small projects. For larger projects, we should probably want to use a proxy server like Nginx or CDN to serve them. Node doesn't work like other servers like Apache or IIS. For Node we are going to have to do the work of opening the file, reading it, and then sending its contents along to the browser.

1) Let's create a directory called ***public***.
2) Inside ***public*** we will create ***home.html, about.html, notfound.html.***

```
Parent dir
    |_public
        |_home.html
        |_about.html
        |_notfound.html
```

3) Then we will create a subdirectory called ***img*** where we will put the image ***logo.jpg***

```
Parent dir
    |_public
        |_img
            |_logo.jpg
        |_home.html
        |_about.html
        |_notfound.html
```

Source: *helloWorld.js*

```
// Importing required packages
var http = require('http')
    Fs = require('fs');

function serveStaticFile(res, path, contentType, responseCode) {
    // make responseCode 200 by default
    if(!responseCode) responseCode = 200;
    // executes the callback function when the file has been read
    fs.readFile(__dirname + path, function(err, data) {
        // If the file doesn't exists
        if(err) {
            res.writeHead(500, { 'Content-Type' : 'text/plain' });
            res.end('500 - Internal Error'); // Scream
        } else {
            res.writeHead(responseCode,
                { 'Content-Type' : contentType });
            res.end(data)
        }
    });
}

// Creates a server that can be listened at port 3000
http.createServer(function(req, res) {
    // normalize url by distilling querystring
```

```
        // trailing slash, and making it lowercase
        var path = req.url.replace(/\/?(?:\?.*)?$/,’’).toLowerCase();
        switch(path) {
                // Redirects to homepage.html
                case ‘’:
                        serveStaticFile(res, ‘/public/home.html’, ‘text/html’);
                        Break;
                // Redirects to about.html
                case ‘/about’:
                        serveStaticFile(res, ‘/public/about.html’, ‘text/html’);
                        Break;
                // Redirects to the logo
                case ‘/img/logo.jpg’:
                        serveStaticFile(res, ‘/public/img/logo.jpg’, ‘image/jpeg’);
                        Break;

                // In the case the urlpath has something dumb like:
        ‘asdalsjldasj’
                // it will return a status code of 404.
                default:
                        serveStaticFile(res, ‘/public/404.html’, ‘text/html’, 404);
                        break;
        }
}).listen(3000);
console.log(‘Server started on localhost:3000; press Ctrl-C to
terminate….’);
```

serveStaticFile is what we call a helper function. It does a lot of the heavy lifting for us.

Code Nuggets

-g (global option) Items installed with this flag are installed in a subdirectory of our Windows home. If you want to use it and your name is two words separated by a space, consider changing it as -g doesn't perform well if there is a space in the username.

Ctrl-S "Freezes" one's computer terminal while…
Ctrl-Q "Unfreezes" the terminal.
npm install <package_name> will install a package and add it to the package.json file.
__dirname will tell you the exact location, or path, of the folder where your code is currently running. It points to the directory where the script you're working on is located.

---

http.createServer(options[, onRequestHandler]) is an inbuilt application programming interface of class http within http module which is used to create a net.Server object.

Parameters:
- **options:** It can be maxDeflateDynamicTableSize, maxSettings, maxSessionMemory, etc according to need.
- **onRequestHandler:** It is an optional parameter passed as parameter of function type.

Return Value: This method returns the object of net.Server.

---

response.writeHead(statusCode[, statusMessage][, headers]) property is an inbuilt property of the 'http' module which sends a response header to the request. The status code is a 3-digit *HTTP* status code, like 404. The last argument, headers, are the response headers. Optionally one can give a human-readable *statusMessage* as the second argument.

Parameters:

- **statusCode** <*number*>**:** It accepts the status codes that are of number type.
- **statusMessage** <*string*>**:** It accepts any string that shows the status message.
- **headers** <*Object*>**:** It accepts any function, array, or string.

Return Value <*http.ServerResponse*>**:** It returns a reference to the *ServerResponse*, so that calls can be chained.

res.end([data] [, encoding]) is used to end the response process.

Parameters: The default encoding is ‘utf8’ and the data parameter is basically the data with which the user wants to end the response.

Return Value: It returns an Object.

IN A NUTSHELL

Node is easy to install in any system. Some systems, like for example Linux, require more preparation to install the correct version of Node.

The use of a terminal, or command prompt, offers significant power and productivity. Popular shells like bash and zsh are available for OS X and Linux. Windows users can opt for Git Bash or explore alternatives like Console, ConEmu, or PowerShell. Virtual machines (VMs) and tools like PuTTY offer alternatives, with Codio providing a Linux instance with an integrated IDE and command line for each project, making it a versatile choice.

Npm is the ubiquitous package manager for Node packages.

Web servers typically deliver static files to our computers for viewing on a browser. However, in Node, the application you create functions as the web server itself. Node offers a framework that simplifies the process of building a web server, providing more control over your application.

Event-Driven Programming is fundamental to Node. To effectively use Node, one must grasp available events and how to respond to them. An illustrative example is the "click event," where the programmer anticipates user button clicks without knowing when they will occur. In Node, handling events, such as HTTP requests, is pivotal. The `http.createServer` method takes a function argument, invoked each time an HTTP request is made, showcasing the event-driven nature of Node.