Table of Content

Installation of Nodejs and Npm

- Windows
  - Installing Nodejs
    - Go to https://nodejs.org/en/download/
    - Download 32/64 bit *Windows Installer(.msi)* depending on your system.
    - Open and run the installer (Administrator rights might be needed)
    - Once installation is complete, open command prompt and execute the command ***node -v***. If node is correctly installed, the version number will show up, for eg. (v6.9.0).
    - NodeJS installation is complete.
  - Installing npm packages
    - Open command prompt
    - Navigate to the folder where you want to install the packages using **cd**
    - Execute ***npm install <package name>***
    - For example ***npm install express***
  - Creating and publishing package with npm
    - Open command prompt and navigate to the folder where you want to create the package.
    - In command prompt, execute **npm init** and follow the instructions. Make sure your module name is unique and (ideally) contains only lowercase letters.
    - A *package.json* file should appear in the folder. Now create and add your script files into the same folder as *package.json*.
    - If you already have an account on npmjs.com, type **npm login**. Else, type **npm adduser**, hit enter and follow the instructions.
    - Use **npm publish** to publish the package. Make sure you update the version before publishing, while publishing future versions using **npm version**
    - Go to www.npmjs.com and search for your package/module. It should be published along with its version and description.

- This package is now public and can be installed by anyone using **npm install <**YourPackageName**>**

- Linux(Ubuntu)

There are several ways to install nodejs on ubuntu.

1. Using ppa

- To install nodejs in linux , first you need to install PPA in order to get access to it's contents. For the latest LTS (the 6.x branch) type the following command in your terminal

  cd
  curl -sL https://deb.nodesource.com/setup_6.x | sudo -E bash -

- Now you can install node.js package:

  sudo apt-get install nodejs

- The nodejs package contains the nodejs binary as well as npm, so you don't need to install npm separately. However, in order for some npm packages to work (such as those that require building from source), you will need to install the build-essentials package, using the following command:

  sudo apt-get install nodejs

2. Alternative method

- Download source code named node-vx.y.z.tar.gz from this link https://nodejs.org/en/download/
- Navigate to the directory where you have this download.
- Extract it using tar

  tar -xvf node-vx.y.z.tar.gz
- For example: tar -xvf node-v.6.9.5.tar.gz
- Run the .configure file and make it

  ./configure
  make
  sudo make install
- Confirm the version of node by typing the following command:

  node -v
- Confirm the npm version as well:

  npm -v
- To install any npm packages use the command:

  npm install <package-name>

Javascript

Javascript Syntax:

Variable Declaration:

```
var number =  23;
var string = “Hello, World!”;
var  array =[“Multiple data types are possible in single array”,12,’hello’,];
```

```
var  objects = {
        “Insert anything here”: “anything here”,
        //you can also use only words without inverted
        //commas
        Like This : 3};
```

- no data type required

Add/Remove Array Item

```
// create an empty array
var myArray = [];

// create array with items. Can store any type
var myOtherArray = [myArray, true, "A random string"];

// call specific value in an array
myOtherArray[0];
// will return myArray

// change value for this item
myOtherArray[0] = false;
// will now return false


// add to end of array
myOtherArray[myOtherArray.length] = "new stuff";
// will return the new item "new stuff"

// or you can use push()
myOtherArray.push("new stuff");
// will return new length of array


// you can remove this last item by using pop()
myOtherArray.pop();
// will return the last item of the array and will have removed it from myOtherArray


// shift and unshift will do the same for the begging of the Array
myOtherArray.shift();
// will remove and return first item of array

myOtherArray.unshift(1,2);
// this will add 1 and 2 to beginning of array and return new length


myOtherArray.unshift(1,2);
// this will add 1 and 2 to beginning of array and return new length
```

```
// you can use delete keyword but turn value to undefine and not shorten length. so we use
splice()
myOtherArray.splice(2, 1);
// this will remove and return  the third item only.
// first arg is where to start and second is how many things to splice. this example is 1.
```

Add/Remove Objects

```
var newObject = {};

// add a property to object
newObject.newPropName = "super fly";

// or other syntax
newObject['other new prop name'] = "mildly fly";

// Now newObject.newPropName will return super fly
newObject.newPropName;

// now to delete
delete newObject.newPropName;
```

Conditionals

```
        // If Else statements
var a = 1;
var b = 2;

if( a < b ) {
 console.log('the if is true!');
} else {
 console.log('the if is false!');
}


// Multi If Else statements
var a = 1;
var b = 2;
var c = 3;

if( a > b ) {
 console.log('the if is true!');
} else if(a > c) {
 console.log('OK, THIS if is Ture!');
} else {
 console.log('None of these were true');
}
```

```
// Ternary operators. same as if else
var a = 1;
var b = 2;

a === b ? console.log('The statement is true') : console.log('The statement is false');

// switch statements
var a = 4;
switch (a) {
 case "Oranges":
        console.log("Orange? really?");
        break;
 case 1:
        console.log("a is equal to 1.");
        break;
 case 2:
        console.log("a is equal to 2.");
        break;
 case 3:
        console.log("a is equal to 3.");
        break;
 case 4:
        console.log("a is equal to 4.");
        break;
 default:
        console.log("I run if no one else is true.");
}
```

Loops

```
// while loop
var i = 0;
while( i < 10 ) {
 console.log(i);
 i += 1
}


// do while loop
var i = 0;
do {
 console.log(i);
 i += 1
} while( i < 10 )


// for loop
for ( var i = 0; i < 10; i++ ) {
  console.log(i);
}

// for in statements
var obj = {a:1, b:2, c:3};
```

```
for ( var prop in obj ) {
 // check if property is inherited or not
 if(obj.hasOwnProperty(prop)) {
        console.log("obj." + prop + " = " + obj[prop]);
 }
}
```

Functions

```
var myFunction = function myFunction(arg1, arg2) {
 console.log(arg1 + " & " + arg2);
        };
```

Timers

```
        function simpleMessage() {
 alert("This is just a simple alert");
}

// set time out
window.setTimeout(simpleMessage, 5000);

// if you wanted to clear the timer.
var timer = window.setTimeout(simpleMessage, 5000);
window.clearTimeout(timer);

// set interval. will repeat every  5000ms
window.setInterval(simpleMessage, 5000);

// if you wanted to clear the intervals.

var intervalHandler = window.setInterval(simpleMessage, 5000);
window.clearInterval(intervalHandle);
```

JSON (Javascript Object Notation)
It is a syntax for storing and exchanging data. When exchanging data between a browser and a server, the data can only be text. Its an data-interchange format, making text exchanged between browsers and servers easier for humans to read and machines to parse and generate. We can convert any JavaScript object into JSON and send JSON to the server. We can also convert any JSON received from the server into JavaScript objects.
A JSON syntax include
Data in key/value pairs
Curly braces hold objects and each name is followed by a semicolon. The key/value pairs are separated by commas
Square brackets hold array and values are separated by commas
Take package.json as an example:

```
{
 "name": "killuabot",
 "version": "1.0.0",
 "description": "I am trying npm",
 "main": "retweet.js",
 "scripts": {
   "start": "node retweet.js"
    "test": "echo \"Error: no test specified\" && exit 1"
 },
 “keywords”:[“demo”],
 "author": "killua",
 "license": "ISC",
 "repository": {
   "type": "git",
   "url": "git+https://github.com/VinitraMk/TwitterBot.git"
 },
 "bugs": {
   "url": "https://github.com/VinitraMk/TwitterBot/issues"
 },
 "homepage": "https://github.com/VinitraMk/TwitterBot#readme"
}
```

You can see that the set of values enclosed inside the curly braces are one object. For example: the set enclosed in the first opening curly braces is one whole object. An object can have several other objects as well. For example the “scripts” object. This object contains 2 key-value pairs,separated by commas. The key is “start” and it’s value is “node retweet.js” The object “keywords” has an array of values enclosed inside square brackets
To access information stored inside the json object we can simply refer to it by the property name I.e key name. For example you can load your json file in an object like this:

```
var jsonobj=require(‘/path/to/package.json’);
```

You can print the value of a property by accessing the key value property in json object with the property name. For example

```
console.log(jsonobj.name);
```

You can iterate over the whole package.json by using for loop

```
for(var objkey in jsonobj){
    console.log(“key: “+ objkey +”value: “+jsonobj.objkey);
}
```

Making a simple date package:

1. Initialize npm package using npm init
2. Enter a unique app name and leave the rest to defaults
3. Edit package.json and add the start in the scripts object as given below

```
//package.json
        {
 "name": "date-package-codecell",
 "version": "1.0.0",
 "description": "A simple package that prints date",
 "main": "index.js",
 "scripts": {
        "test": "echo \"Error: no test specified\" && exit 1",
        "start": "node index.js"
        //add this line
 },
 "keywords": [
        "nodejs.package",
        "tutorial",
        "workshop"
 ],
 "author": "Chaitya Shah",
 "license": "ISC"
}
```

4. Make a new file and name it “index.js”

   Add the following code in it.

```
        //index.js
        var date = new Date();
        console.log(date.getDate()+"/"+date.getMonth()+1+"/"+(date.getYear()
-100
        //for printing time instead of date
        /*
                var  today = new Date();
                var h = today.getHours();
                var m = today.getMinutes();
                var s = today.getSeconds();

        */
));
```

5. Save the file and now login into your npmjs account using “npm login”.

6. Publish the package with the help of “npm publish”

7. Read made package “npm install date-package-codecell” (Only for reference)

Twitter bot with Node.JS using twit(package).

1. **Installing ‘twit’ package using npm**

Suppose you are developing some network application which uses some kind of api, and all you have with you is your javascript file. You might want to use someone else’s library of code in your application. Nodejs, being open source has a wide community of

developers, who contribute awesome packages to improve the functionality of applications, developed on nodejs.
The packages are published by the nodejs contributors and you can use these packages using
NPM i.e Node Package Manager.
One such package that we will be using for the Twitter bot is the 'twit' package
Install the twit package by typing the following command:
npm install twit
It is preferable to use the command: npm install twit --save
Adding --save as an argument to npm install means saving a reference of this package to your package.json file
Using this save argument will create a directory named node_modules in your home directory. This directory stores all the npm packages that you install. So in future you can use this directory to run the code from other packages.

2. Setting up a Twitter app

- Create an empty directory and initialize it with npm init.
- Create a javascript file in this directory. For example config.js. You can think of it as a configurations files which will contain your authorization keys, to grant access to your twitter app.
- Now go to https://apps.twitter.com/
- Click on the "Create New App" button
- Fill up the mandatory application details like the application name, description of application and also provide your website url if you have one. If you don't, just enter a fake url.
- Check the developer agreement checkbox and click on "Create your twitter application".
- After creating the application, look for "Keys and Access Tokens" under nav-panes and click on "Generate Access Tokens". Then copy the Consumer Key,Consumer Secret,Access Token and Access Token Secret into the config.js file.

```
module.exports = {
consumer_key: '',
consumer_secret: '',
access_token: '',
access_token_secret: ''
}
```

- Nodejs allows you to store code in a file, which you might want to reference in another file. It's like creating your own mini-package for your own code. So by using module.exports you create an object which contains all the keys required.

3. Building a Bot

Once you are done saving the config.js, we can write some code

- Create a new file. For example bot.js
- Open the file bot.js and type the following statement:

```
var tweet=require('twit');
```

- The require() method works like an import statement. In nodejs it is used to import modules and files that you wish to use in your application.
- Using the require method load the ‘twit’ module and also the config file like given below:

```
var tweet=require(‘twit’);
var config=require(‘./config’);
```

- Pass the configuration of your twitter application i.e your access tones and keys, to a twitter object as shown below:

```
var Twitter=new twit(config);
```

- Now comes a code which will find the latest tweet and retweet it
- We will start with initializing the params object which sets the properties to search the tweets based on queries. The “q” property i.e the query property will filter the searches. You can feed a # hashtag to this property so your bot searches tweets based on the #hashtag you have provided and retweet one of them.
- This is how we start the retweet function

```
var retweet=function(){
    var params={
        q:’#nodejs’,    //Required
        result_type:’recent’,
        lang:’en’ }
```

- The property result_type tells the bot to search only for the latest tweets. The other two properties are optional, to filter your search even more.
- We will now use the Twitter.get function to search for the tweets based on parameters in params. The Twitter.get function is provided by ‘twit’ API to GET any of the RESTful API endpoints. Endpoints are basically what you will make your http client to point at, to interact with the data resources.
- It has 3 parameters: API endpoint, params object and a callback.
- A callback function is basically a function which is called after a particular task is executed.
- The code for retweeting is give below

```
// find latest tweet according the query 'q' in params
var retweet = function() {
  var params = {
    q: '#POTUS44, #POTUS',  // REQUIRED
    result_type: 'recent',
    lang: 'en'
  }
  Twitter.get('search/tweets', params, function(err, data) {
   // if there no errors
    if (!err) {
     // grab ID of tweet to retweet
      var retweetId = data.statuses[0].id_str;
      // Tell TWITTER to retweet
      Twitter.post('statuses/retweet/:id', {
        id: retweetId
      }, function(err, response) {
        if (response) {
          console.log('Retweeted!!!');
```

```
            }
            // if there was an error while tweeting
            if (err) {
                console.log('Something went wrong while RETWEETING... Duplication
maybe...');
            }
        });
    }
    // if unable to Search a tweet
    else {
      console.log('Something went wrong while SEARCHING...');
    }
  });
}
```

- In the above code the get method will search for tweets based on the properties set inside the params object and after the task is done a callback function is called. This function takes two parameters: The error object and the response object.
- If no error is produced the id of the latest tweet is read. Then the post method is called with the params object containing the id of the latest tweet and another callback function is called. If no error is produced then on getting response the latest tweet will be retweeted again. You can set the interval for retweet using the setInterval() method and automate this action at specific intervals.
- We can also mark favorite a random tweet, which is similar to retweeting a tweet. The difference here is the tweet selected will be random and not a fixed pre-determined tweet. Similar to retweet, we will start by creating a param object setting up the queries for search. Then we will call the same get function that we did, while writing the retweeting code. We will obtain the status of the search and pass it to a function to pick a random tweet from the array of statuses.
- We will use the same post method that we used for retweeting, except now we will use ‘favorites/create’ endpoint in the post method. Similar to what we did in retweeting we can automate this process by using setInterval() method. The gist for marking favorite tweets is given below:

```
// find a random tweet and 'favorite' it
var favoriteTweet = function(){
 var params = {
    q: '#POTUS, #POTUS44',  // REQUIRED
    result_type: 'recent',
    lang: 'en'
 }
 // find the tweet
 Twitter.get('search/tweets', params, function(err,data){

   // find tweets
   var tweet = data.statuses;
   var randomTweet = ranDom(tweet);   // pick a random tweet

   // if random tweet exists
   if(typeof randomTweet != 'undefined'){
     // Tell TWITTER to 'favorite'
     Twitter.post('favorites/create', {id: randomTweet.id_str}, function(err, response){
       // if there was an error while 'favorite'
       if(err){
```

```
        console.log('CANNOT BE FAVORITE... Error');
      }
      else{
        console.log('FAVORITED... Success!!!');
      }
    });
  }
 });
}
// grab & 'favorite' as soon as program is running...
favoriteTweet();
// 'favorite' a tweet in every 60 minutes
setInterval(favoriteTweet, 7200000);

// function to generate a random tweet tweet
function ranDom (arr) {
 var index = Math.floor(Math.random()*arr.length);
 return arr[index];
};
```