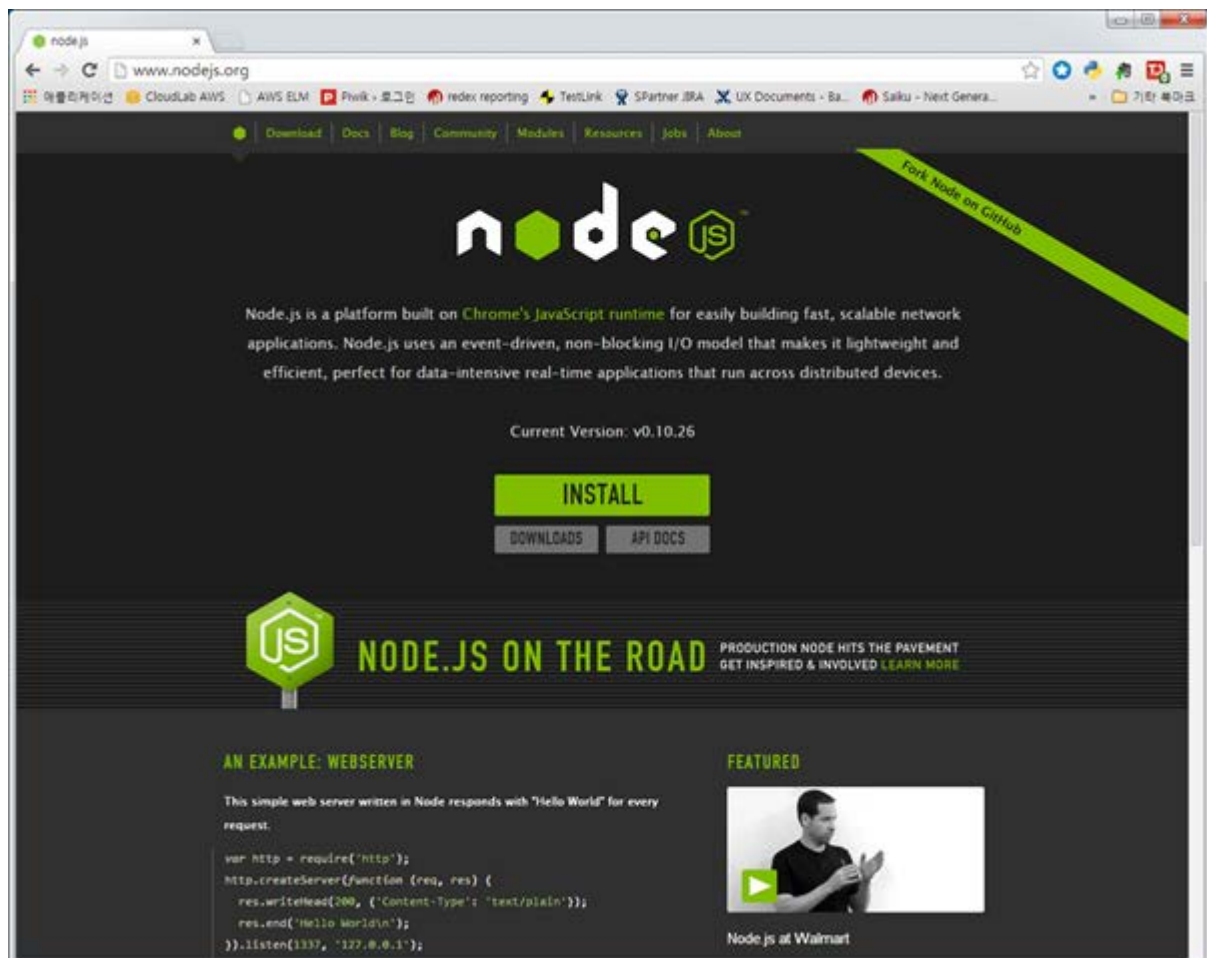


아두이노와 node.js 통신 환경 구축하기

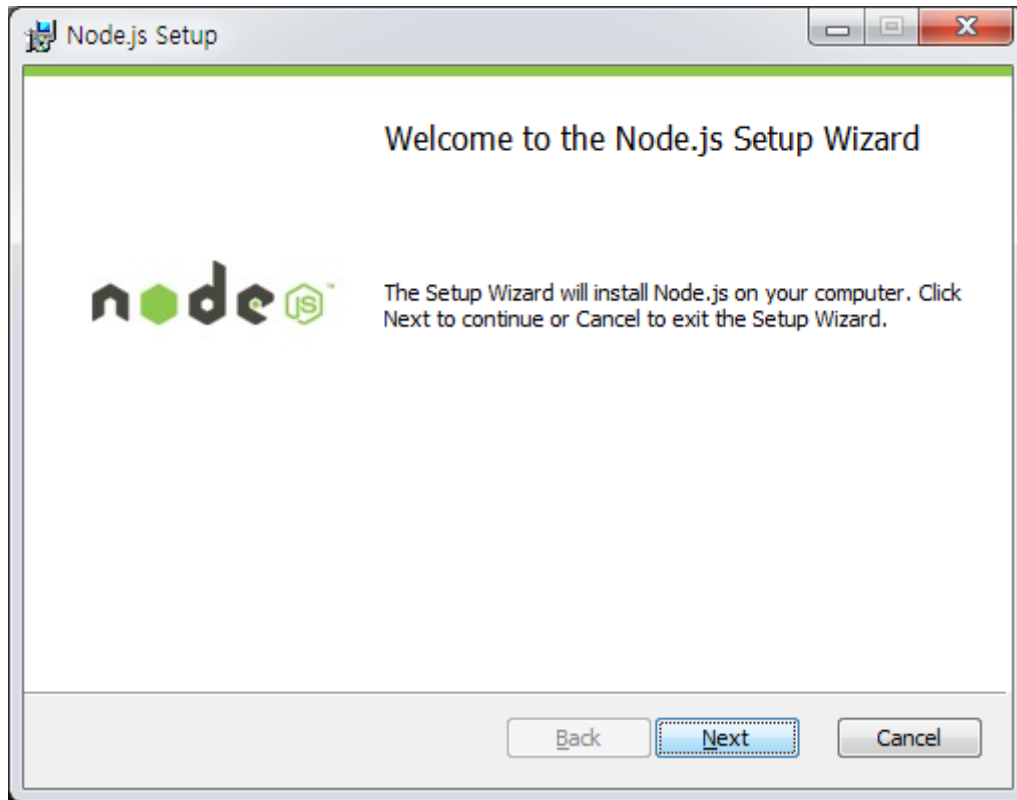
Node.js 설치하고 개발환경 설정하기

다운로드 하기

http://www.nodejs.org 페이지에서 install 버튼을 누르면 OS 에 맞는 인스톨러를 다운로드 해준다.



다음으로 installer 를 실행한다.



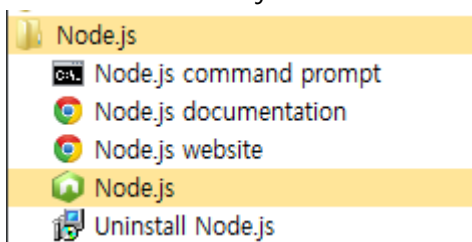
이제 node.js 가 설치되었는지 확인하자. 시작버튼을 누른다.



모든 프로그램을 눌러 보자.

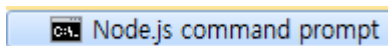
▶ 모든 프로그램

다음과 같이 Node.js 관련 파일들이 설치된 것을 볼 수 있다.

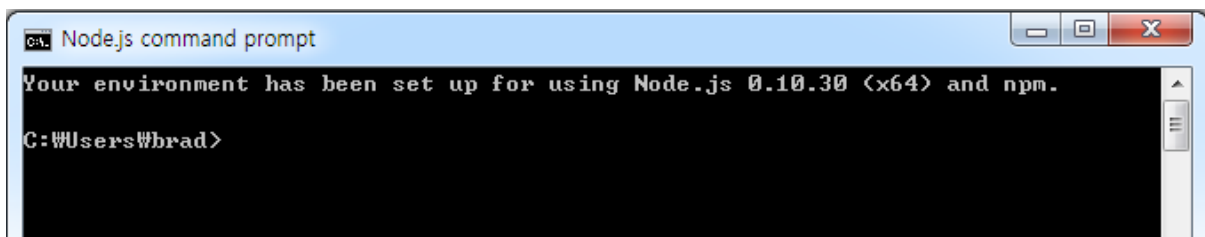


설치를 끝냈으면 이제 간단한 웹서버를 만들어보자. <http://www.nodejs.org> 홈페이지에 있는 예제를 이용한다.

먼저 다음 프로그램을 실행시킨다.



다음과 같이 실행된다.



AN EXAMPLE: WEBSERVER

This simple web server written in Node responds with "Hello World" for every request.

```
var http = require('http');
http.createServer(function (req, res) {
  res.writeHead(200, {'Content-Type': 'text/plain'});
  res.end('Hello World\n');
}).listen(1337, '127.0.0.1');
console.log('Server running at http://127.0.0.1:1337/');
```

To run the server, put the code into a file `example.js` and execute it with the `node` program from the command line:

```
% node example.js
Server running at http://127.0.0.1:1337/
```

Here is an example of a simple TCP server which listens on port 1337 and echoes whatever you send it:

```
var net = require('net');

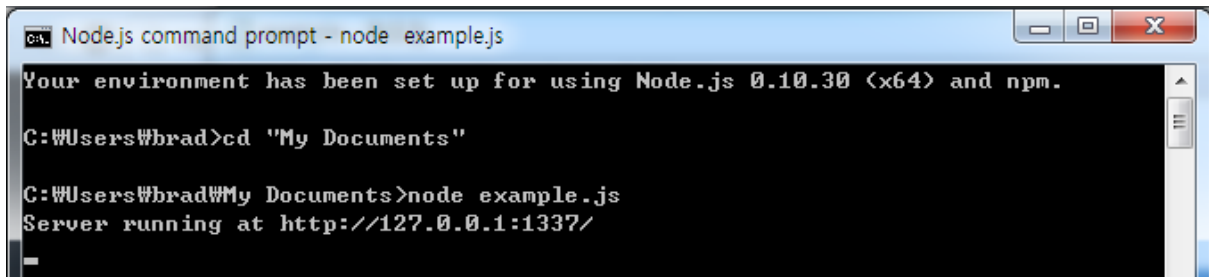
var server = net.createServer(function (socket) {
  socket.write('Echo server\r\n');
  socket.pipe(socket);
});

server.listen(1337, '127.0.0.1');
```

노트장을 열어 다음과 같이 작성한다.

```
var http = require('http');
http.createServer(function (req, res) {
  res.writeHead(200, {'Content-Type': 'text/plain'});
  res.end('Hello World\n');
}).listen(1337, '127.0.0.1');
console.log('Server running at http://127.0.0.1:1337/');
```

이 코드를 `example.js` 로 저장한다. 여기서는 [내 문서]에 저장한다.
다음으로 다음과 같이 해당 파일을 실행한다.

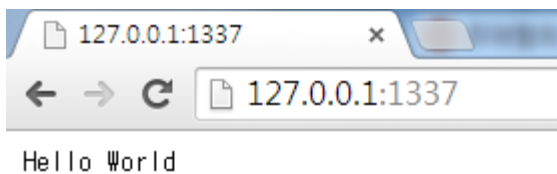


```
Node.js command prompt - node example.js
Your environment has been set up for using Node.js 0.10.30 <x64> and npm.

C:\Users\Wbrad>cd "My Documents"

C:\Users\Wbrad\My Documents>node example.js
Server running at http://127.0.0.1:1337/
```

이제 웹브라우저로 확인해보면 다음과 같이 메시지가 출력되는 것을 확인할 수 있다.

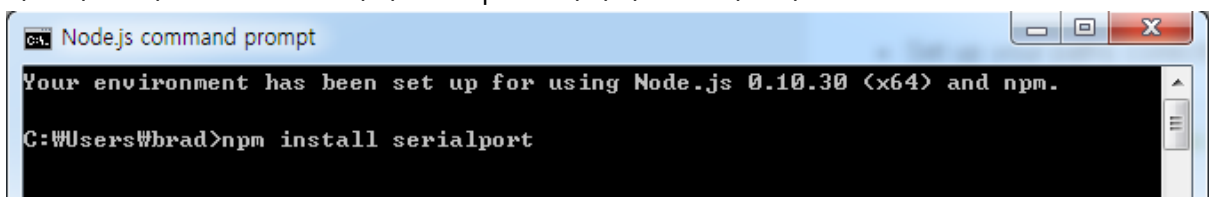


이제 node.js 를 이용하여 시리얼 포트를 제어해 보자. node.js 를 이용하여 시리얼 포트를 제어하기 위해서는 serialport 패키지를 설치해야 한다.

다음 사이트를 참조한다.

<https://www.npmjs.org/package/serialport>

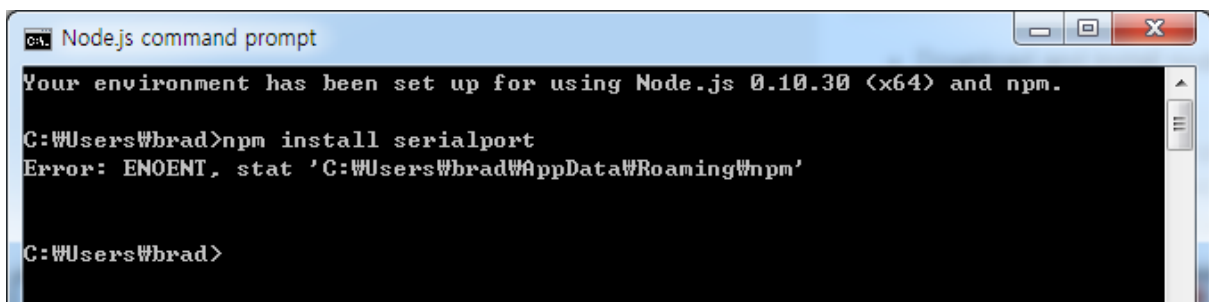
다음과 같이 명령을 실행시켜 serialport 패키지를 설치한다.



```
Node.js command prompt
Your environment has been set up for using Node.js 0.10.30 <x64> and npm.

C:\Users\Wbrad>npm install serialport
```

그런데 다음과 같이 에러 메시지가 발생한다.



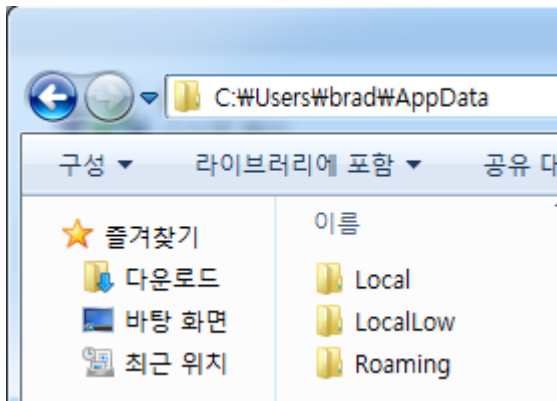
```
Node.js command prompt
Your environment has been set up for using Node.js 0.10.30 <x64> and npm.

C:\Users\Wbrad>npm install serialport
Error: ENOENT, stat 'C:\Users\Wbrad\AppData\Roaming\npm'

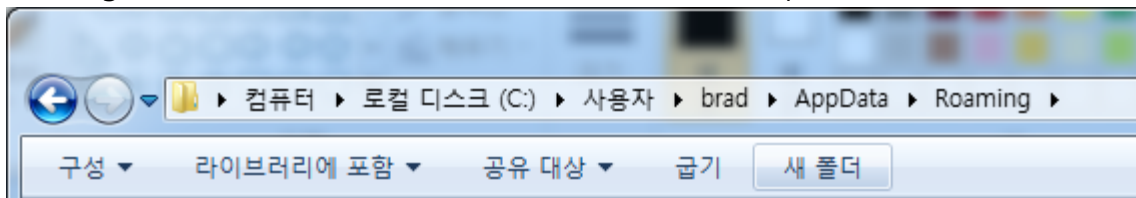
C:\Users\Wbrad>
```

npm 디렉터리가 C:\Users\사용자\AppData\Roaming 디렉터리에 만들어지지 않아 발생하는 메시지이다. 일단 C:\Users\사용자 디렉터리로 이동한다. 그런데 AppData

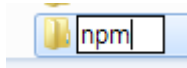
디렉터리가 보이지 않는다. 다음과 같이 직접 AppData 를 입력하면 디렉터리 내용이 보이게 된다.



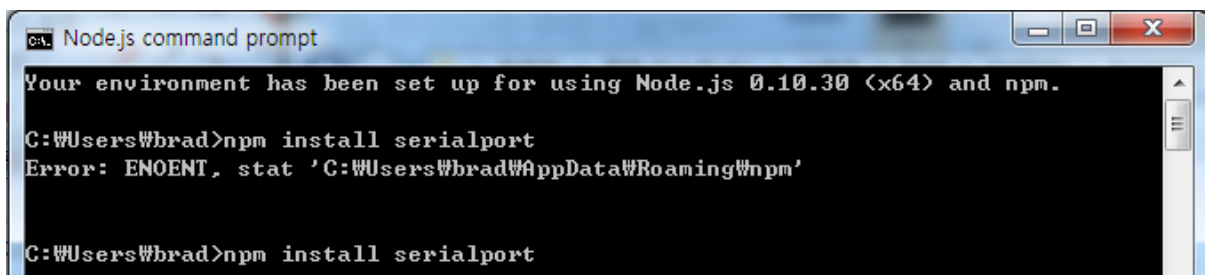
Roaming 디렉터리로 들어가 [새 폴더] 메뉴를 이용하여 npm 폴더를 만들어준다.



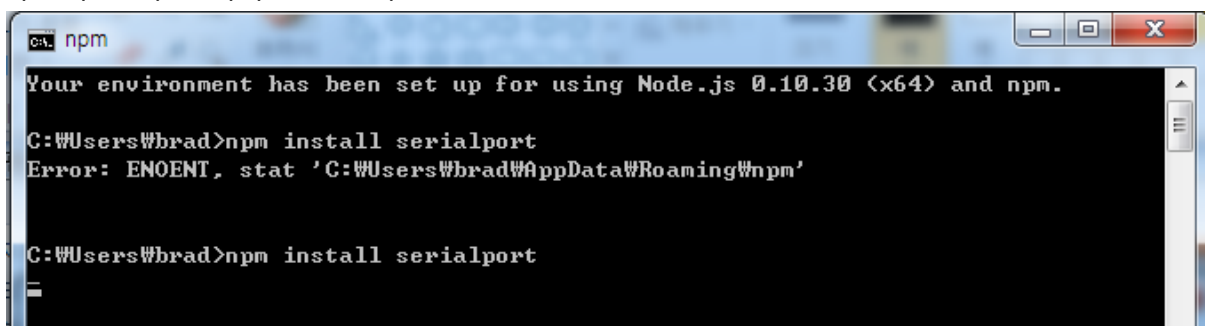
다음과 같이 만들어준다.



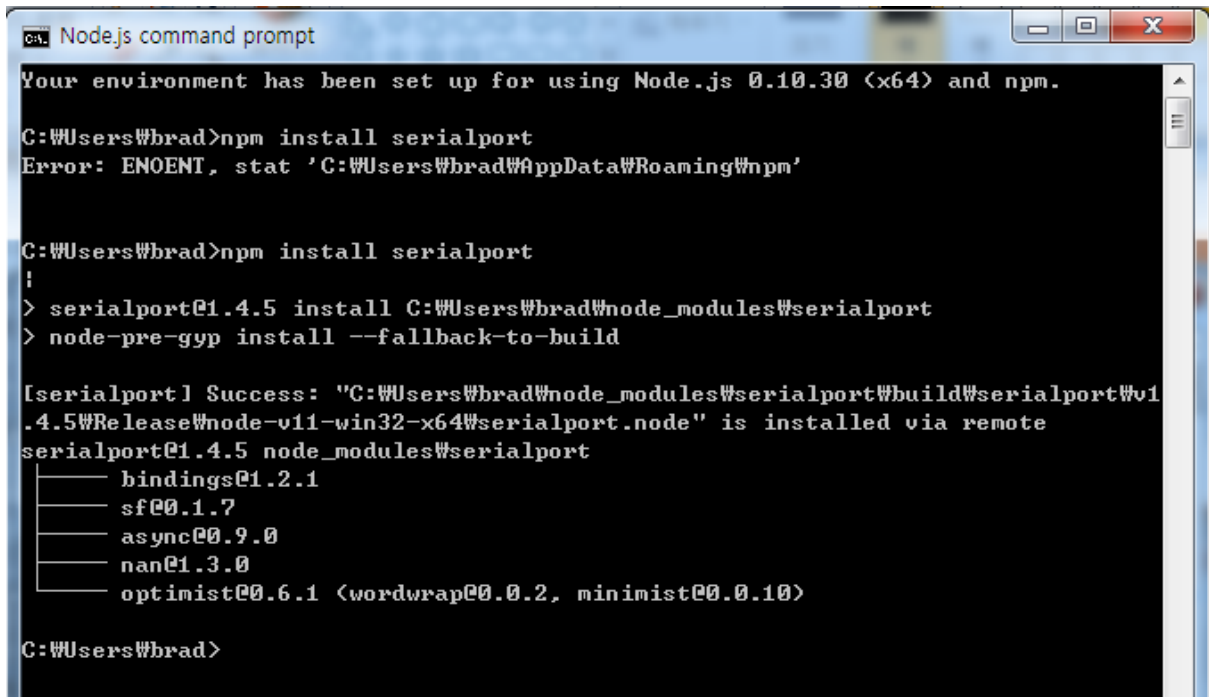
다시 한번 다음과 같이 명령을 주도록 하자.



다음과 같이 설치가 진행된다.



다음과 같이 설치가 완료된다.



```
Node.js command prompt
Your environment has been set up for using Node.js 0.10.30 (x64) and npm.

C:\Users\brad>npm install serialport
Error: ENOENT, stat 'C:\Users\brad\AppData\Roaming\npm'

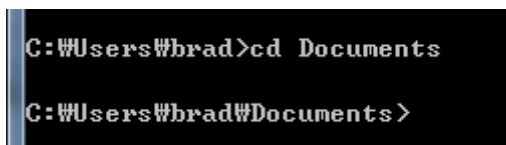
C:\Users\brad>npm install serialport
!
> serialport@1.4.5 install C:\Users\brad\node_modules\serialport
> node-pre-gyp install --fallback-to-build

[serialport] Success: "C:\Users\brad\node_modules\serialport\build\serialport\v1.4.5\Release\node-v11-win32-x64\serialport.node" is installed via remote
serialport@1.4.5 node_modules\serialport
├── bindings@1.2.1
├── sf@0.1.7
├── async@0.9.0
├── nan@1.3.0
└── optimist@0.6.1 (wordwrap@0.0.2, minimist@0.0.10)

C:\Users\brad>
```

이제 설치된 serialport 패키지를 테스트해 보자.

다음과 같이 Documents 디렉터리로 이동한다.



```
C:\Users\brad>cd Documents
C:\Users\brad\Documents>
```

노트장을 열어 다음과 같이 작성한다.

```
var serialPort = require("serialport");

serialPort.list(function (err, ports) {
  ports.forEach(function(port) {
    console.log(port.comName);

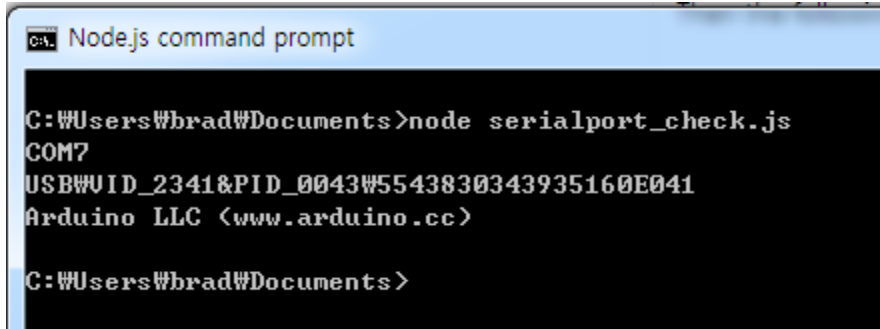
    console.log(port.pnpId);

    console.log(port.manufacturer);
  });
});
```

이 코드를 serialport_check.js 로 저장한다. 여기서는 [내 문서]에 저장한다.

아두이노가 컴퓨터에 연결된 것을 확인한다.

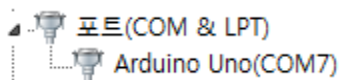
실행시키면 다음과 같이 출력된다.

A screenshot of a Node.js command prompt window. The title bar says "Node.js command prompt". The command prompt shows the following text:

```
C:\Users\brad\Documents>node serialport_check.js
COM7
USB\VID_2341&PID_0043\5543830343935160E041
Arduino LLC <www.arduino.cc>
C:\Users\brad\Documents>
```

여기서는 COM7 로 표시된다.

장치 관리자에서 다음과 같이 포트를 확인한다.



이제 serialport 패키지를 사용해 보자.

아두이노 스케치를 다음과 같이 작성한 후, 업로드한다.

```
void setup() {
  Serial.begin(9600);
}

void loop() {
  Serial.print("Hello node.js");

  delay(1000);
}
```

노트장을 열어 다음과 같이 작성한다.

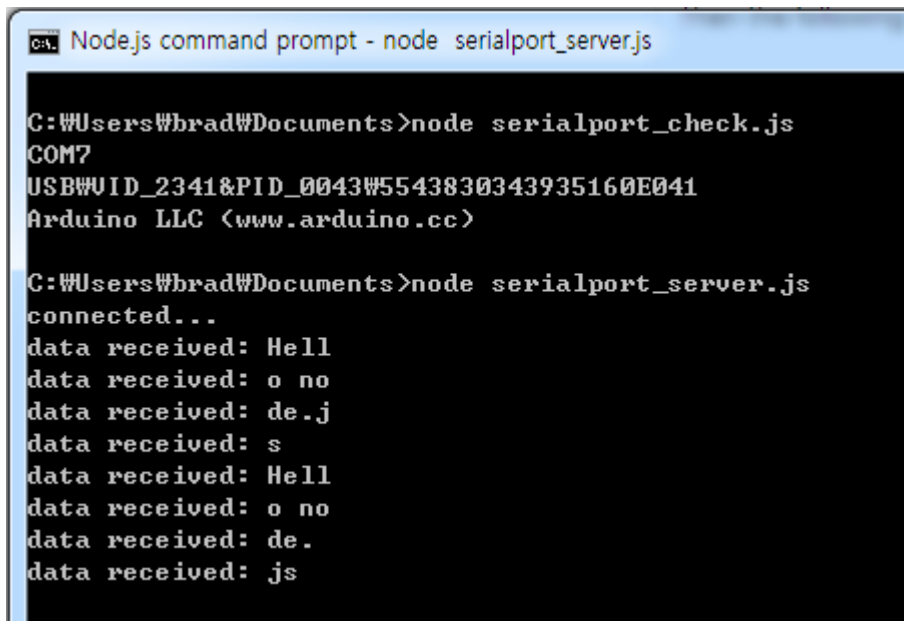
```
var SerialPort = require("serialport").SerialPort
var serialPort = new SerialPort("COM7", {
  baudrate: 9600
}, false);
```



```
serialPort.open(function () {  
    console.log('connected...');  
    serialPort.on('data', function(data) {  
        // 아두이노에서 오는 데이터를 출력한다.  
        console.log('data received: ' + data);  
    });  
});
```

이 코드를 serialport_server.js 로 저장한다. 여기서는 [내 문서]에 저장한다.

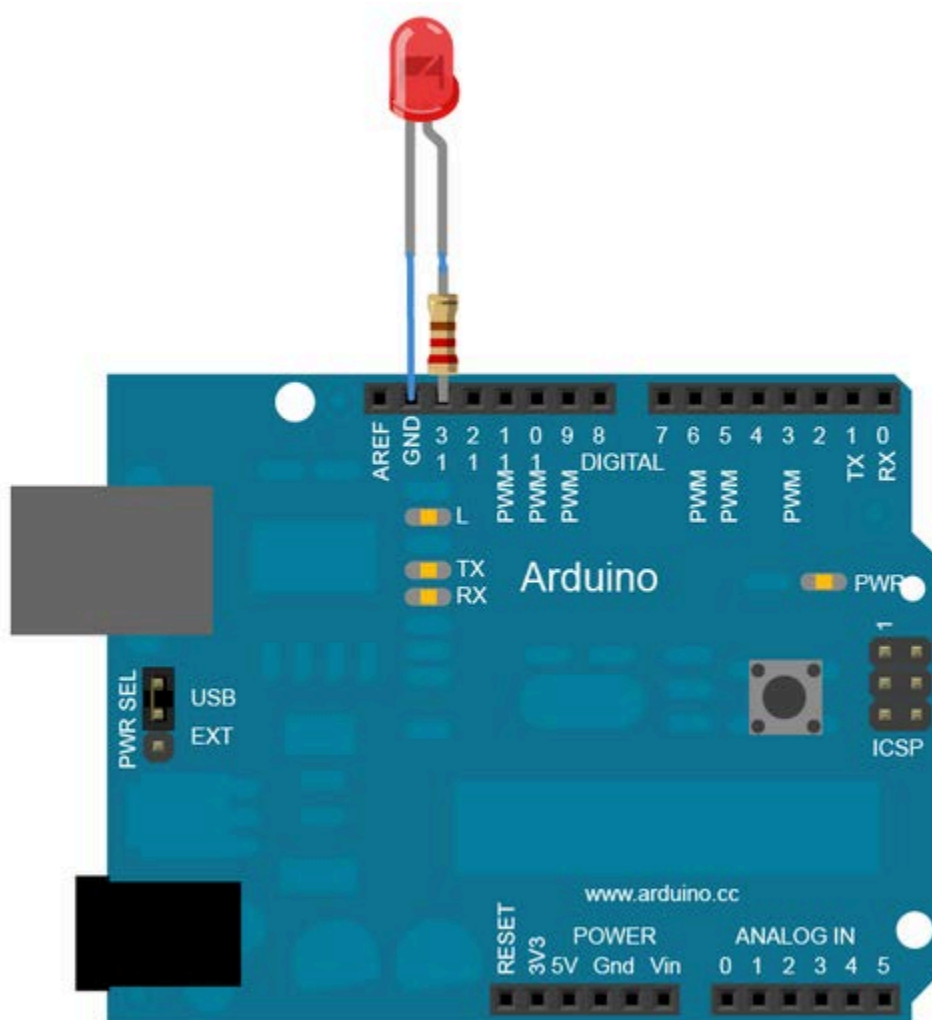
실행시키면 다음과 같이 출력된다.



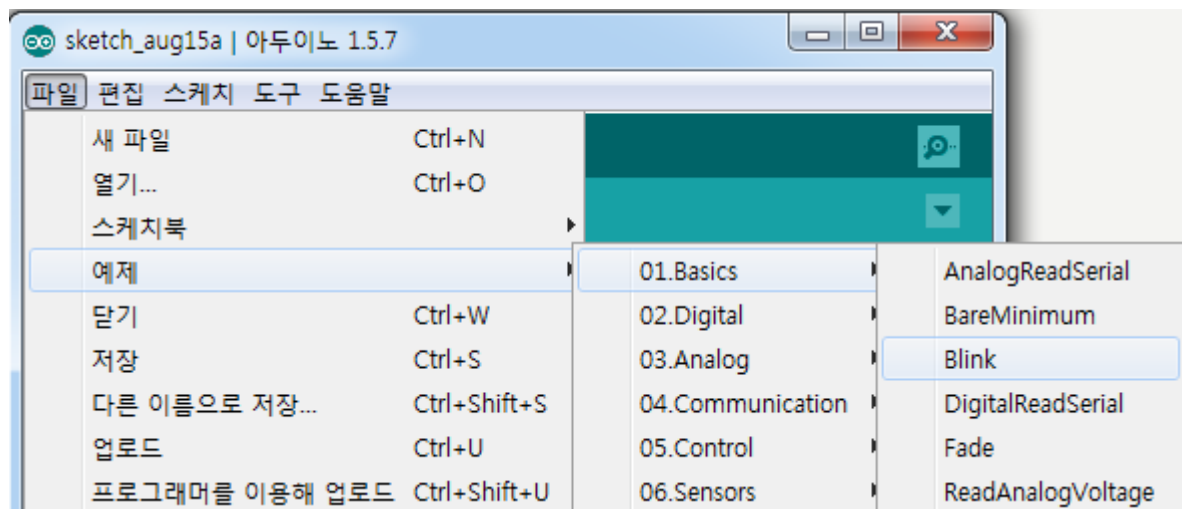
```
Node.js command prompt - node serialport_server.js  
  
C:\Users\brad\Documents>node serialport_check.js  
COM7  
USB\VID_2341&PID_0043\5543830343935160E041  
Arduino LLC (www.arduino.cc)  
  
C:\Users\brad\Documents>node serialport_server.js  
connected...  
data received: Hell  
data received: o no  
data received: de.  
data received: s  
data received: Hell  
data received: o no  
data received: de.  
data received: js
```

이제 node.js 를 이용하여 아두이노의 LED 를 제어해 보자.

LED 를 다음과 같이 연결한다.



다음과 같이 Blink 예제를 연다.



컴파일하고 업로드하면 LED가 깜빡이는 것을 볼 수 있다.
예제를 다음과 같이 수정한다.

```
/*
  Blink
  Turns on an LED on for one second, then off for one second, repeatedly.
  This example code is in the public domain.
*/

// Pin 13 has an LED connected on most Arduino boards.
// give it a name:
int led = 13;

// the setup routine runs once when you press reset:
void setup() {
  // initialize the digital pin as an output.
  Serial.begin(9600);
  pinMode(led, OUTPUT);
}

// the loop routine runs over and over again forever:
void loop() {
  static int incomingValue = 0;           // 보낸값을 저장하기 위한 변수 선언

  if ( Serial.available() > 0 ) { // 뭔가 입력값이 있다면
    incomingValue = Serial.read();
  }

  if ( incomingValue == 49 ) { // 값이 '1' 이면
    digitalWrite(13, HIGH); // LED를 켜다.
  }

  if ( incomingValue == 48 ) { // 값이 '0' 이면
    digitalWrite(13, LOW); // LED를 끈다.
  }
  Serial.print(incomingValue);
}
```

컴파일하고 업로드 한다.

노트장을 열어 다음과 같이 작성한다.

```
var SerialPort = require("serialport").SerialPort
// Arduino 가 "COM7" 에 연결되었다고 가정한다.
var serialPort = new SerialPort("COM7", { baudrate: 9600}, false);
var ledStatus = 0;

serialPort.open(function () {
  console.log('connected...');
  serialPort.on('data', function(data) {
    // 아두이노에서 오는 데이터를 출력한다.
    console.log('data received: ' + data);
  });
  setInterval(function(){
    ledStatus = !ledStatus;
    console.log(ledStatus);
    // LED 가 ON/OFF 된다.
    serialPort.write(ledStatus==true ? "1" : "0", function(err, results) {});
  }, 1000);
});
```

이 코드를 serialport_server_led_on_off.js 로 저장한다. 여기서는 [내 문서]에 저장한다.

serialport_server_led_on_off.js 파일을 다음과 같이 실행시킨다.

```
>node serialport_server_led_on_off.js
```

LED가 주기적으로 깜빡이는 것을 볼 수 있다.

이제 NodeJS로 아주 간단한 웹서버를 띄운 후 웹에서 LED 켜보자.

노트장을 열어 다음과 같이 작성한다.

```
var serialport = require("serialport");
var http = require('http');
var UrlParser = require('url');

var serialPort = new serialport.SerialPort( "COM7", { baudrate : 9600});

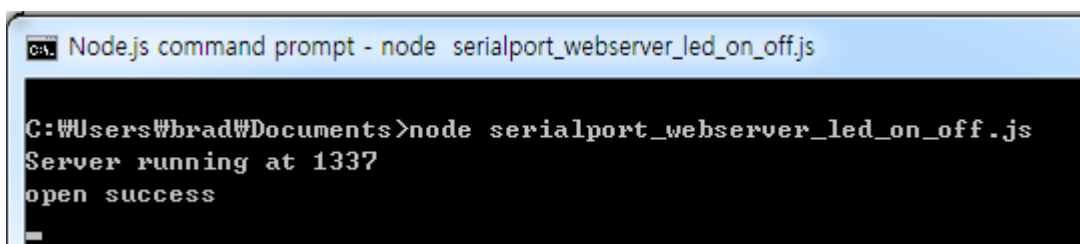
serialPort.on("open",function() {
    console.log("open success")
});

http.createServer(function (req, res) {
    var result = UrlParser.parse(req.url,true);
    if(result.pathname == '/on') {
        serialPort.write("1",function() { });
    }
    else if(result.pathname == '/off') {
        serialPort.write("0",function() { });
    }

    res.writeHead(200, {'Content-Type': 'text/plain'});
    res.end('Hello Arduino\n');
}).listen(1337);
console.log('Server running at 1337');
```

이 코드를 serialport_webserver_led_on_off.js 로 저장한다. 여기서는 [내 문서]에 저장한다.

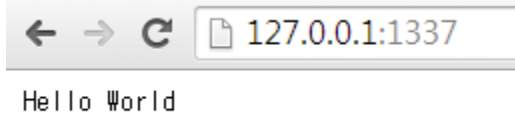
serialport_webserver_led_on_off.js 파일을 실행시킨다. 간단한 웹서버가 구동된다.



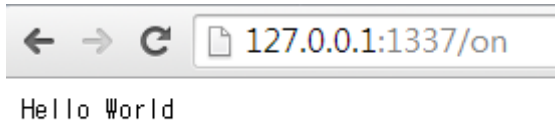
```
Node.js command prompt - node serialport_webserver_led_on_off.js

C:\Users\brad\Documents>node serialport_webserver_led_on_off.js
Server running at 1337
open success
```

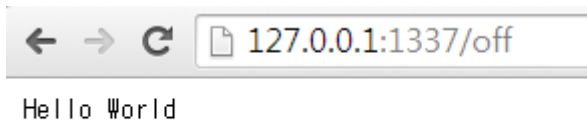
다음과 같이 웹브라우저로 서버에 접속한다.



다음과 같이 on을 붙인다. LED가 켜지는 것을 볼 수 있다.



다음과 같이 off로 변경해 본다. LED가 꺼지는 것을 볼 수 있다.



그러면 이제 Express 라는 NodeJS 모듈을 이용해 간단히 LED 를 제어해 보도록 하자.
express 는 웹 프레임워크로 이것을 이용하면 웹서버를 손쉽게 구현할 수 있다.

Express 에 대해 간단히 살펴보면 다음과 같다.

node.js 는 여러 종류의 웹 개발 프레임워크를 제공한다. 얼마전에 Paypal 이 내부 시스템을 대규모로 node.js 로 전환하면서 오픈 소스화한 KarkenJS 나 Meteo 등 여러가지 프레임워크가 있는데, 그 중에서 가장 많이 사용되는 프레임워크 중하나인 Express 에 대해서 설명하고자 한다.

Express 는 웹 페이지 개발 및 REST API 개발에 최적화된 프레임워크로 매우 사용하기가 쉽다.

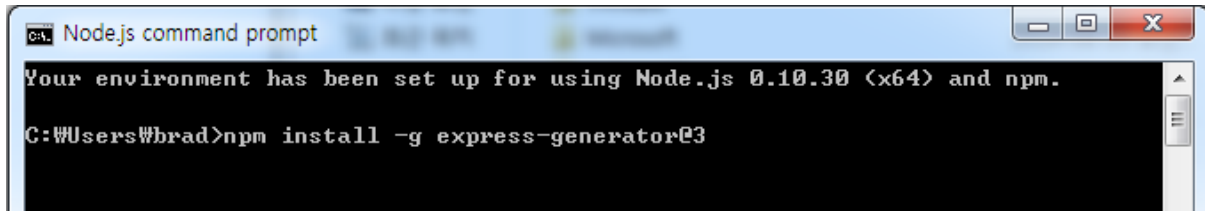
Express 를 설치하기 위해서는 다음 사이트에 있는 Quick Start 를 참고한다.

<https://www.npmjs.org/package/express-generator>

그러면 express 를 설치해 보자.

다음과 같이 express-generator를 설치한다. 이렇게 하면

사용자계정\AppData\Roaming\npm\node_modules\express-generator\bin에 javascript로 작성된 express 파일을 생성한다. 이후부터 console에서 express를 실행할 수 있다.



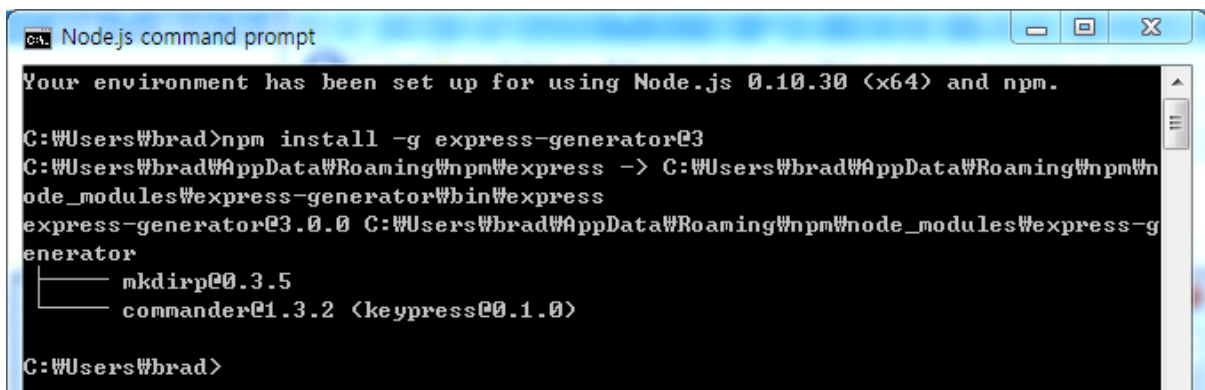
```
C:\> Node.js command prompt

Your environment has been set up for using Node.js 0.10.30 <x64> and npm.

C:\Users\brad>npm install -g express-generator@3
```

마지막에 @3 은 express-generator 버전을 나타낸다. windows 에서는 이전 버전을 사용해야 한다.

다음과 같이 설치가 완료된다.



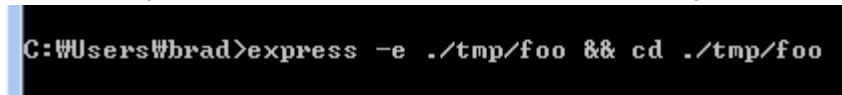
```
C:\> Node.js command prompt

Your environment has been set up for using Node.js 0.10.30 <x64> and npm.

C:\Users\brad>npm install -g express-generator@3
C:\Users\brad\AppData\Roaming\npm\express -> C:\Users\brad\AppData\Roaming\npm\node_modules\express-generator\bin\express
express-generator@3.0.0 C:\Users\brad\AppData\Roaming\npm\node_modules\express-generator
├── mkdirp@0.3.5
└── commander@1.3.2 <keypress@0.1.0>

C:\Users\brad>
```

다음과 같이 프로젝트를 생성하고 해당 디렉터리로 이동한다. -e 옵션은 웹 프레임워크 엔진으로 ejs 를 사용한다는 의미이다. 기본 엔진은 jade 이다.



```
C:\Users\brad>express -e ./tmp/foo && cd ./tmp/foo
```

여기서 프로젝트 디렉터리가 꼭 ./tmp/foo 일 필요는 없다. 예를 들어 다음과 같이 명령을 줄 수도 있다.

```
>express -e ./my_webserver && cd ./my_webserver
```

다음과 같이 설치된다.

```
C:\Users\Wbrad>express -e ./tmp/foo && cd ./tmp/foo

create : ./tmp/foo
create : ./tmp/foo/package.json
create : ./tmp/foo/app.js
create : ./tmp/foo/public/javascripts
create : ./tmp/foo/public/images
create : ./tmp/foo/public/stylesheets
create : ./tmp/foo/public/stylesheets/style.css
create : ./tmp/foo/routes
create : ./tmp/foo/routes/index.js
create : ./tmp/foo/routes/user.js
create : ./tmp/foo/views
create : ./tmp/foo/views/index.ejs
create : ./tmp/foo/views/error.ejs
create : ./tmp/foo/public
create : ./tmp/foo/bin
create : ./tmp/foo/bin/www

install dependencies:
$ cd ./tmp/foo && npm install

run the app:
$ DEBUG=my-application ./bin/www

C:\Users\Wbrad\Wtmp\Wfoo>
```

다음과 같이 디렉터리의 내용을 확인해 보자.

```
C:\Users\Wbrad\Wtmp\Wfoo>dir
C 드라이브의 볼륨에는 이름이 없습니다.
볼륨 일련 번호: 7C52-27D0

C:\Users\Wbrad\Wtmp\Wfoo 디렉터리

2014-08-16 오후 10:55 <DIR> .
2014-08-16 오후 10:55 <DIR> ..
2014-08-16 오후 10:55      1,357 app.js
2014-08-16 오후 10:55 <DIR> bin
2014-08-16 오후 10:55      332 package.json
2014-08-16 오후 10:55 <DIR> public
2014-08-16 오후 10:55 <DIR> routes
2014-08-16 오후 10:55 <DIR> views
                2개 파일                1,689 바이트
                6개 디렉터리 82,845,696,000 바이트 남음
```

다음과 같이 의존성이 있는 추가 패키지를 설치한다.

```
C:\Users\Wbrad\Wtmp\Wfoo>npm install
```


다음과 같이 추가 패키지가 설치된다.

```
C:\Users\brad\tmp\foo>npm install
npm WARN deprecated static-favicon@1.0.2: use serve-favicon module
debug@0.7.4 node_modules\debug

static-favicon@1.0.2 node_modules\static-favicon

ejs@0.8.8 node_modules\ejs

cookie-parser@1.0.1 node_modules\cookie-parser
├── cookie-signature@1.0.3
└── cookie@0.1.0

morgan@1.0.1 node_modules\morgan
└── bytes@0.3.0

body-parser@1.0.2 node_modules\body-parser
├── qs@0.6.6
├── raw-body@1.1.7 <string_decoder@0.10.25-1, bytes@1.0.0>
└── type-is@1.1.0 <mime@1.2.11>

express@3.4.8 node_modules\express
├── methods@0.1.0
├── merge-descriptors@0.0.1
├── range-parser@0.0.4
├── cookie-signature@1.0.1
├── fresh@0.2.0
├── buffer-crc32@0.2.1
├── cookie@0.1.0
├── mkdirp@0.3.5
├── send@0.1.4 <mime@1.2.11>
├── commander@1.3.2 <keypress@0.1.0>
├── connect@2.12.0 <uid2@0.0.3, qs@0.6.6, pause@0.0.1, bytes@0.2.1, raw-body@
1.1.2, hatch@0.5.0, negotiator@0.3.0, multiparty@2.2.0>
└── ...
C:\Users\brad\tmp\foo>
```

다음과 같이 추가로 설치된 패키지를 확인해 보자.

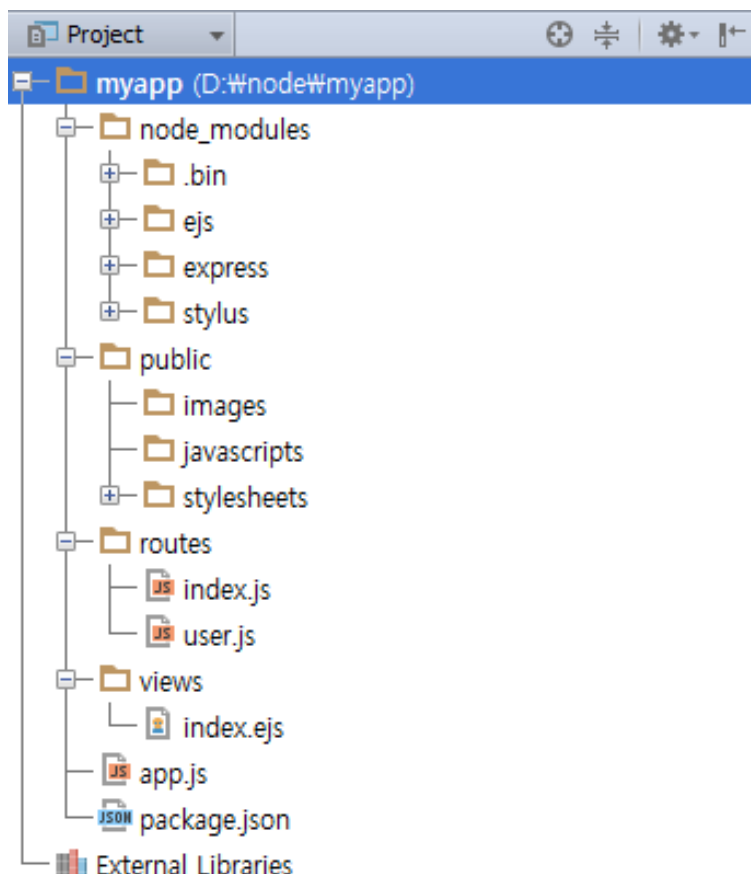
```
C:\Users\brad\tmp\foo>dir
C 드라이브의 볼륨에는 이름이 없습니다.
볼륨 일련 번호: 7C52-27D0

C:\Users\brad\tmp\foo 디렉터리

2014-08-16 오후 10:57 <DIR> .
2014-08-16 오후 10:57 <DIR> ..
2014-08-16 오후 10:55      1,357 app.js
2014-08-16 오후 10:55 <DIR> bin
2014-08-16 오후 10:57 <DIR> node_modules
2014-08-16 오후 10:55      332 package.json
2014-08-16 오후 10:55 <DIR> public
2014-08-16 오후 10:55 <DIR> routes
2014-08-16 오후 10:55 <DIR> views
                2개 파일                1,689 바이트
                7개 디렉터리 82,830,393,344 바이트 남음
```

node_modules 디렉터리가 추가된 것을 볼 수 있다.

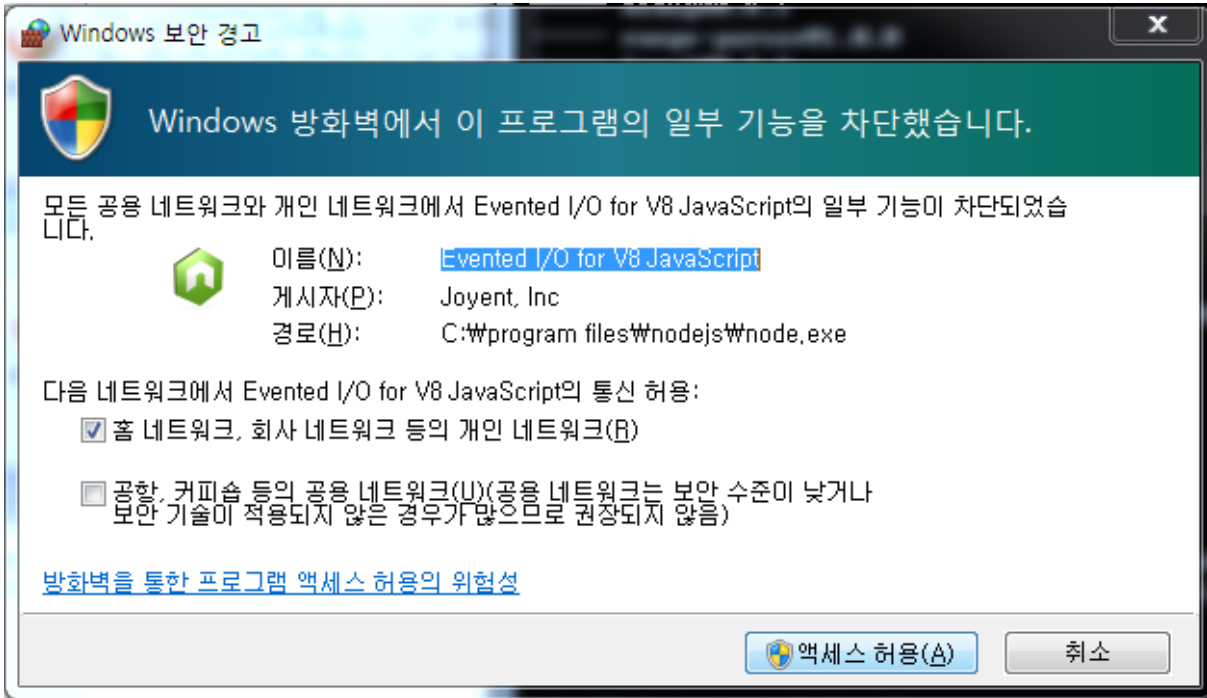
다음은 지금까지의 디렉터리 구조이다.



이제 다음과 같이 웹서버를 구동시킨다.

```
C:\Users\brad\tmp\foo>npm start
```

다음 창이 뜨면 [액세스 허용] 버튼을 누른다.

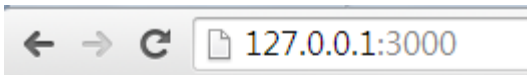


그러면 다음과 같이 웹서버가 구동되는 것을 볼 수 있다.

```
C:\Users\brad\tmp\foo>npm start

> application-name@0.0.1 start C:\Users\brad\tmp\foo
> node ./bin/www
```

다음과 같이 3000번 포트로 접속한다. 웹서버가 동작하는 것을 볼 수 있다.



Express

Welcome to Express

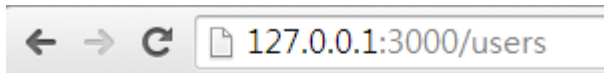
콘솔창도 다음과 같이 나타나는 것을 확인한다.

```
C:\Users\brad\tmp\foo>npm start

> application-name@0.0.1 start C:\Users\brad\tmp\foo
> node ./bin/www

GET / 200 20ms - 207b
GET /stylesheets/style.css 200 3ms - 110b
```

다음과 같이 접속해 보자.



respond with a resource

콘솔창도 다음과 같이 확인하자.

```
C:\Users\brad\tmp\foo>npm start

> application-name@0.0.1 start C:\Users\brad\tmp\foo
> node ./bin/www

GET / 200 20ms - 207b
GET /stylesheets/style.css 200 3ms - 110b
GET /users 304 3ms
```

그러면 어떻게 웹서버가 구동되는지 살펴보자.

먼저

npm start 하면

npm이 package.json을 찾아 "start" script를 찾아 읽는다.

그러면 node ./bin/www 명령에 의해 ./bin/www 파일이 읽힌다.

./bin/www 파일은 상위 디렉터리에 있는 app.js 파일을 읽는다.

npm이 start script를 찾지 못하면 node server.js를 수행한다.

사용자가 http://127.0.0.1:3000 주소로 접근하면

views/index.ejs 파일의 내용을 보내준다.

package.json

```
{
  "name": "application-name",
  "version": "0.0.1",
  "private": true,
```

```
"scripts": {  
  "start": "node ./bin/www"  
},  
"dependencies": {  
  "express": "~3.4.8",  
  "static-favicon": "~1.0.0",  
  "morgan": "~1.0.0",  
  "cookie-parser": "~1.0.1",  
  "body-parser": "~1.0.0",  
  "debug": "~0.7.4",  
  "ejs": "~0.8.5"  
}  
}
```

bin/www 파일의 내용을 확인해 본다.

```
#!/usr/bin/env node  
var debug = require('debug')('my-application');  
var app = require('./app');  
  
app.set('port', process.env.PORT || 3000);  
  
var server = app.listen(app.get('port'), function() {  
  debug('Express server listening on port ' + server.address().port);  
});
```

app.js 파일의 내용은 다음과 같다.

```
var express = require('express');  
var http = require('http');  
var path = require('path');  
var favicon = require('static-favicon');  
var logger = require('morgan');  
var cookieParser = require('cookie-parser');  
var bodyParser = require('body-parser');  
  
var routes = require('./routes');  
var users = require('./routes/user');
```

```

var app = express();

// view engine setup
app.set('views', path.join(__dirname, 'views'));
app.set('view engine', 'ejs');

app.use(favicon());
app.use(logger('dev'));
app.use(bodyParser.json());
app.use(bodyParser.urlencoded());
app.use(cookieParser());
app.use(express.static(path.join(__dirname, 'public')));
app.use(app.router);

app.get('/', routes.index);
app.get('/users', users.list);

/// catch 404 and forwarding to error handler
app.use(function(req, res, next) {
    var err = new Error('Not Found');
    err.status = 404;
    next(err);
});

/// error handlers

// development error handler
// will print stacktrace
if (app.get('env') === 'development') {
    app.use(function(err, req, res, next) {
        res.render('error', {
            message: err.message,
            error: err
        });
    });
}

```

```

}

// production error handler
// no stacktraces leaked to user
app.use(function(err, req, res, next) {
  res.render('error', {
    message: err.message,
    error: {}
  });
});

module.exports = app;

```

require는 필요한 module들을 로드하는 역할을 한다.

node.js에서 router는 특정 URL로 들어오는 HTTP Request에 대한 handler(node에서는 router라고 한다)를 의미한다.

다음 부분을 살펴보자.

```

var routes = require('./routes');
var users = require('./routes/user');

```

여기서 routes 핸들러는 ./routes/index.js 파일을 의미하고 users 핸들러는 ./routes/user.js 파일을 의미한다. index.js 파일은 디폴트 핸들러로 생각한다.

다음과 같이 명령을 수행해 보자.



```

C:\Users\brad\tmp\foo>dir routes
C 드라이브의 볼륨에는 이름이 없습니다.
볼륨 일련 번호: 7C52-27D0

C:\Users\brad\tmp\foo\routes 디렉터리

2014-08-16 오후 10:55 <DIR> .
2014-08-16 오후 10:55 <DIR> ..
2014-08-16 오후 10:55 105 index.js
2014-08-16 오후 10:55 102 user.js
                2개 파일                207 바이트
                2개 디렉터리 82,830,135,296 바이트 남음

```

./routes 디렉터리 아래에 두 개의 파일이 있는 것을 볼 수 있다.

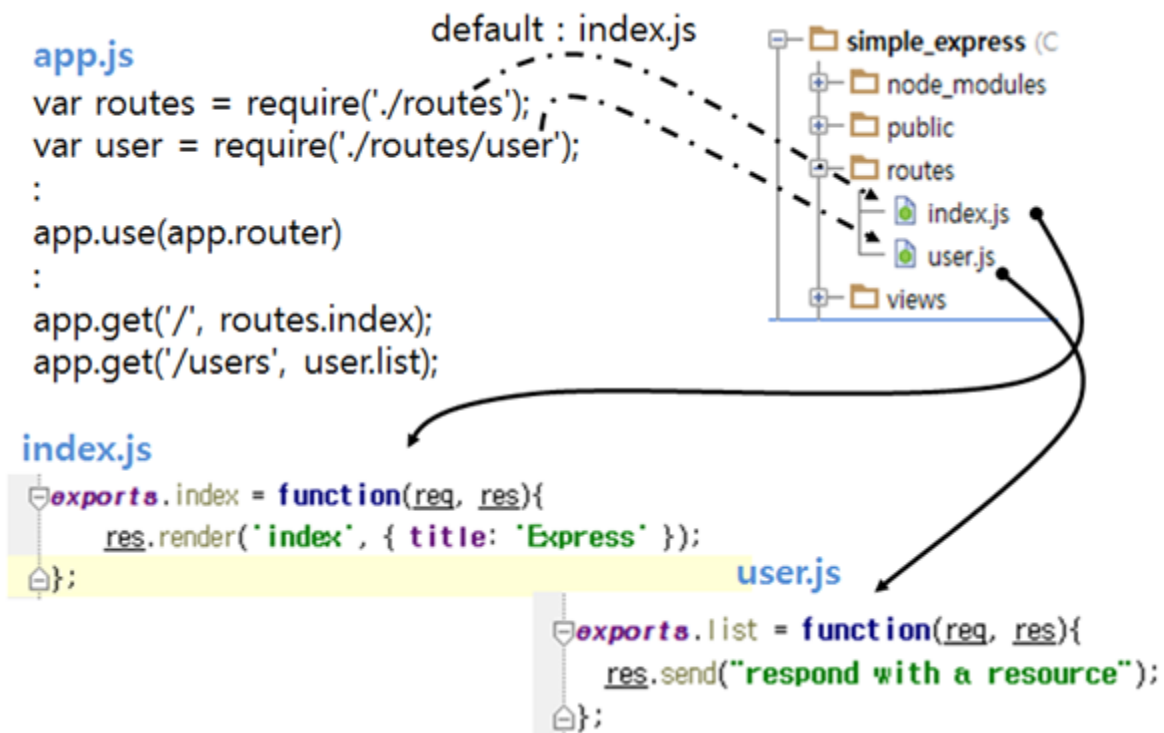
다음 부분을 살펴보자.

```
app.get('/', routes.index);
```

```
app.get('/users', users.list);
```

여기서 웹서버의 /(웹서버의 기본 홈페이지 위치)에 대한 HTTP GET 요청이 들어오면 routes.index 핸들러를 사용하여 처리한다는 의미이다. 즉, ./routes 디렉터리에 있는 index.js 파일을 읽어 index 함수를 이용하여 처리한다. /users 디렉터리에 대한 HTTP GET 요청이 들어오면 ./routes 디렉터리에 있는 users.js 파일을 읽어 list 함수를 이용하여 처리한다는 의미이다.

다음 그림을 참조한다.



다음은 routes/index.js 파일의 내용이다.

routes/index.js

```
/* GET home page. */
exports.index = function(req, res){
  res.render('index', { title: 'Express' });
};
```

다음은 routes/users.js 파일의 내용이다.

```
/* GET users listing. */
exports.list = function(req, res){
  res.send('respond with a resource');
};
```

이번엔 routes/index.js 파일에 의해 어떤 파일이 사용되는지 살펴보자. 먼저 다음 부분의 의미를 알아보자.

```
// view engine setup
app.set('views', path.join(__dirname, 'views'));
app.set('view engine', 'ejs');
```

여기서는 템플릿 엔진이 ejs로 지정되어 있다. 그리고 ejs 템플릿 파일을 저장할 위치를 `__dirname/views`로 지정되어 있는 것을 볼 수 있다. 템플릿 파일의 위치를 `path.join(__dirname, 'view')`로 정의하였는데, `__dirname`은 프로그램이 현재 수행 중인 파일의 위치, 즉, `app.js`가 위치한 디렉터리를 의미하며, `path.join`을 이용하면, `${현재디렉터리}/views`라는 경로로 지정한 것이다.

`routes/index.js` 파일의 내용을 보면, `request`에 대해 `rendering`을 할 때, `index`라는 템플릿을 부르고(앞에서 엔진과 `views` 디렉터리를 지정했기 때문에, `__dirname/views/index.ejs` 파일을 사용하게 된다. 이 때 인자로 `title="Express"` 변수를 넘기게 된다.

HTTP response로 응답을 보내는 방법을 `rendering`이라고 하는데, 간단한 문자열의 경우, `response.send("문자열");` 을 이용해서 보낼 수 도 있다. 또는 `response code`를 싫어서 보낼 때는 `response.send(404, "페이지를 찾을 수 없습니다.");` 와 같은 식으로 첫 번째 인자에 HTTP response code를 실어서 보내는 것도 가능하다.

그러면, `views/index.ejs` 파일의 내용을 살펴보자.

다음은 `views/index.ejs` 파일의 내용이다.

```
<!DOCTYPE html>
<html>
  <head>
    <title><%= title %></title>
    <link rel='stylesheet' href='/stylesheets/style.css' />
  </head>
  <body>
    <h1><%= title %></h1>
    <p>Welcome to <%= title %></p>
  </body>
</html>
```

일반적인 HTML과 거의 유사하다. Parameter를 사용하고자 할 때는 ASP나 JSP처럼 `<%= 변수%>` 로 사용하면 된다. 마찬가지로, `for`, `while`, `if`등 간단한 스크립트 로직도 작성할 수 있다.

좀 더 자세한 내용은 다음 사이트를 참조한다.

<http://bcho.tistory.com/887>

그러면 예제를 수정해서 LED가 Toggle되는 간단한 프로젝트를 만들어 보도록 하자.

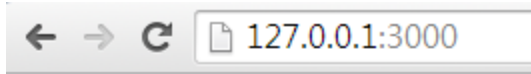
먼저 views/index.ejs 파일을 다음과 같이 수정한다.

```
<!DOCTYPE html>
<html>
  <head>
    <title><%= title %></title>
    <link rel='stylesheet' href='/stylesheets/style.css' />
  </head>
  <body>
    <h1><%= title %></h1>
    <p>Welcome to <%= title %></p>

    <!-- 간단히 버튼으로 POST를 보내도록 한다. -->
    <form action="/" method="post">
      <button type="submit">눌러봐</button>
    </form>

  </body>
</html>
```

웹을 실행시켜보자.



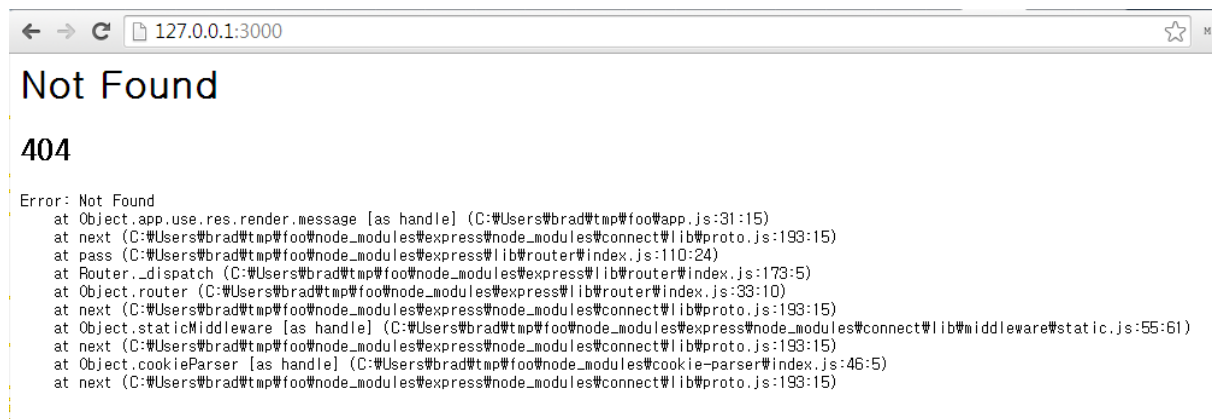
Express

Welcome to Express

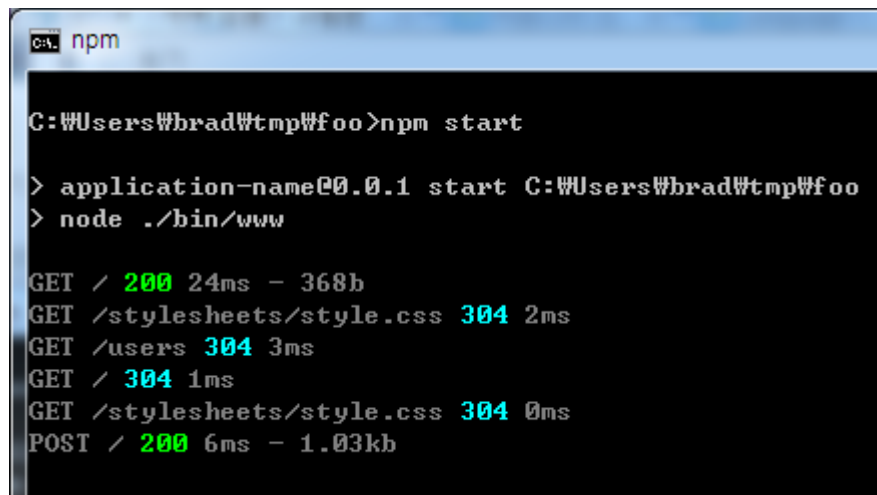
눌러봐

버튼이 나타나는 것을 볼 수 있다.

버튼을 누르면 다음과 같이 에러 메시지가 뜬다.



콘솔창도 확인하자.



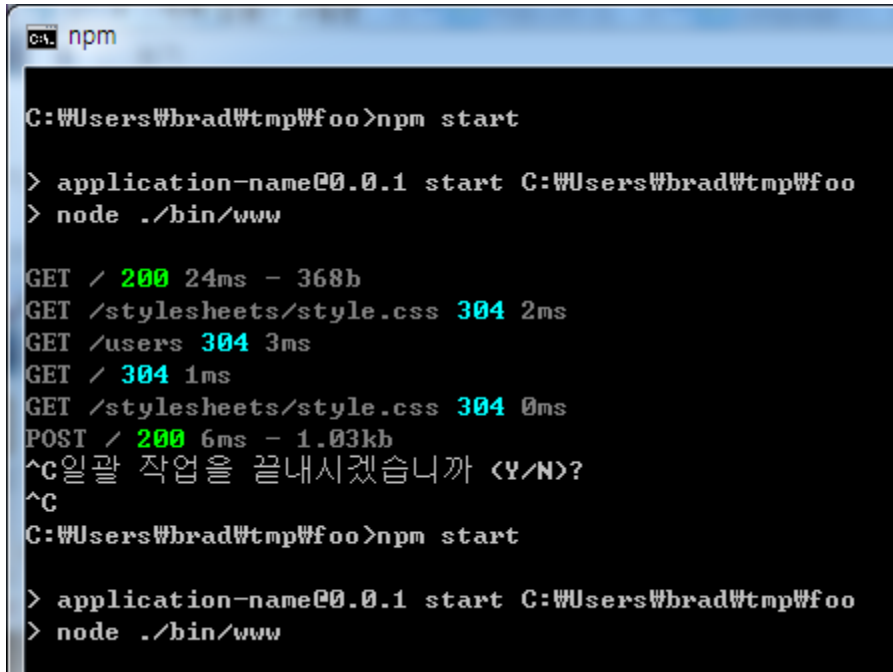
그러면 app.js 파일을 수정해 보자.

다음 부분을 app.js 파일의 마지막에 추가한다.

```
//  
// 이 값에 따라 LED 켜지는게 결정된다.  
//  
var ledStatus = false;  
//  
// POST 액션을 받으면 LED가 Toggle 되도록 한다.  
//  
app.post('/', function(req, res) {  
  ledStatus = ledStatus ? false : true;  
  res.render('index', { title: 'Express' });  
  console.log(ledStatus);  
});
```

});

ctrl+c 키를 두 번 눌러 프로그램을 종료한다. 그리고 서버를 다시 구동시킨다. 다음 그림을 참조한다.



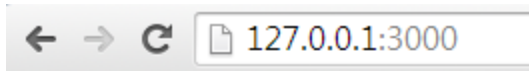
```
C:\Users\brad\tmp\foo>npm start

> application-name@0.0.1 start C:\Users\brad\tmp\foo
> node ./bin/www

GET / 200 24ms - 368b
GET /stylesheets/style.css 304 2ms
GET /users 304 3ms
GET / 304 1ms
GET /stylesheets/style.css 304 0ms
POST / 200 6ms - 1.03kb
^C일괄 작업을 끝내시겠습니까 <Y/N>?
^C
C:\Users\brad\tmp\foo>npm start

> application-name@0.0.1 start C:\Users\brad\tmp\foo
> node ./bin/www
```

그리고 버튼을 연속으로 눌러본다.



Express

Welcome to Express

눌러봐

그러면 다음과 같이 표시된다. 여기서는 버튼을 네 번 누른 경우의 화면이다.

```
C:\Users\brad\tmp\foo>npm start
> application-name@0.0.1 start C:\Users\brad\tmp\foo
> node ./bin/www

GET / 200 31ms - 368b
GET /stylesheets/style.css 304 4ms
POST / 200 3ms - 368b
true
GET /stylesheets/style.css 304 5ms
POST / 200 1ms - 368b
false
GET /stylesheets/style.css 304 1ms
POST / 200 2ms - 368b
true
GET /stylesheets/style.css 304 7ms
POST / 200 1ms - 368b
false
GET /stylesheets/style.css 304 0ms
```

그러면 ledStatus 값을 시리얼포트로 보내 LED를 제어해 보자.

다음 내용을 app.js 파일의 뒷부분에 추가한다.

```
//
// Serial Port를 사용하는 예제
//
var SerialPort = require("serialport").SerialPort

// Arduino가 "COM7" 에 연결되었다고 가정한다.
var serialPort = new SerialPort("COM7", {baudrate: 9600}, false);

serialPort.open(function () {
  console.log('접속되었습니다!');
  serialPort.on('data', function(data) {
    // Arduino에서 오는 데이터를 출력한다.
    console.log('data received: ' + data);
  });
  setInterval(function(){
    // LED가 ON/OFF 될꺼예요
    serialPort.write(ledStatus ? "1" : "0", function(err, results) {});
  }, 100);
});
```

파일을 저장하고 서버를 재구동시키자. 그리고 버튼을 눌러보자. LED가 버튼을 반복적으로 누름에 따라 꺼졌다 켜졌다 하는 것을 볼 수 있다.

다음과 같이 소스를 수정해 보자. 먼저 `app.post()` 함수에 다음을 추가한다.

```
app.post('/', function(req, res) {
  ledStatus = ledStatus ? false : true;
  res.render('index', { title: 'Express' });
  console.log(ledStatus);
  serialPort.write(ledStatus ? "1" : "0", function(err, results) {});
});
```

그리고 `setInterval()` 함수를 호출하는 부분을 주석처리한다.

```
serialPort.open(function () {
  console.log('접속되었습니다!');
  serialPort.on('data', function(data) {
    // Arduino에서 오는 데이터를 출력한다.
    console.log('data received: ' + data);
  });
  /*
  setInterval(function(){
    // LED가 ON/OFF 될꺼예요
    serialPort.write(ledStatus ? "1" : "0", function(err, results) {});
    }, 100);
  */
});
```

함수의 순서는 그대로 두어도 상관없다.

파일을 저장하고 서버를 재구동시키자. 그리고 버튼을 눌러보자. LED가 버튼을 반복적으로 누름에 따라 꺼졌다 켜졌다 하는 것을 볼 수 있다.

이상 express 모듈을 이용해 LED를 제어해 보았다.

매번 스케치를 업로드 하지 않고 스크립트처럼 쓰고자 할 경우엔 Firmata를 사용한다. Firmata를 사용하면 매번 업로드 하지 않고도 Arduino를 자유롭게 제어할 수 있다.

Firmata? 뭐죠?

Firmata는 아두이노 같은 마이크로 컨트롤러와 PC가 서로 통신하기 위해 고안된 범용 프로토콜입니다.

어떻게 쓰는건가요?

일단, Arduino에 최초 StandardFirmata 라는 스케치를 업로드 합니다. 그 후에, Arduino와 통신을 하기 위해서는 Firmata 프로토콜에 맞도록 메시지만 보내주시면 됩니다.

만약, NodeJS를 사용해서 Arduino와 통신하고 싶다면 NodeJS으로 제작된 Firmata 모듈을 사용해 Arduino를 제어할 수 있습니다.

NodeJS를 예로 들어볼까요?

NodeJS 쪽 Firmata 모듈 중 Johnny-Five라는 Firmata based Arduino Framework이 있습니다. 다음과 같은 형태로 사용할 수 있습니다.

```
//
// Johnny-five 모듈로 보드에 접속해 LED를 제어한다.
//
var five = require('johnny-five')
  , board = new five.Board();

board.on("ready", function() {
  //
  // 13번 PIN을 OUTPUT으로
  //
  this.pinMode(13, 1);

  //
  // 0.1s마다 돌면서 ledStatus 값에 따라 LED를 ON/OFF 한다.
  //
```

```
this.loop(100, function() {  
    //  
    this.digitalWrite( 13, ledStatus ? 1 : 0 );  
});  
});
```

위 예제를 이전에 만들었던 app.js 마지막 부분에 붙입니다. 이전에 붙였던 serialport 부분은 떼어냅니다.

다음은 수정된 app.js 파일의 내용이다.

```
var express = require('express');  
var http = require('http');  
var path = require('path');  
var favicon = require('static-favicon');  
var logger = require('morgan');  
var cookieParser = require('cookie-parser');  
var bodyParser = require('body-parser');  
  
var routes = require('./routes');  
var users = require('./routes/user');  
  
var app = express();  
  
// view engine setup  
app.set('views', path.join(__dirname, 'views'));  
app.set('view engine', 'ejs');  
  
app.use(favicon());  
app.use(logger('dev'));  
app.use(bodyParser.json());  
app.use(bodyParser.urlencoded());  
app.use(cookieParser());  
app.use(express.static(path.join(__dirname, 'public')));  
app.use(app.router);  
  
app.get('/', routes.index);
```

```

app.get('/users', users.list);

/// catch 404 and forwarding to error handler
app.use(function(req, res, next) {
    var err = new Error('Not Found');
    err.status = 404;
    next(err);
});

/// error handlers

// development error handler
// will print stacktrace
if (app.get('env') === 'development') {
    app.use(function(err, req, res, next) {
        res.render('error', {
            message: err.message,
            error: err
        });
    });
}

// production error handler
// no stacktraces leaked to user
app.use(function(err, req, res, next) {
    res.render('error', {
        message: err.message,
        error: {}
    });
});

module.exports = app;

//
// 이 값에 따라 LED 켜지는게 결정된다.
//

```

```

var ledStatus = false;
//
// POST 액션을 받으면 LED가 Toggle 되도록 한다.
//
app.post('/', function(req, res) {
  ledStatus = ledStatus ? false : true;
  res.render('index', { title: 'Express' });
  console.log(ledStatus);
  // serialPort.write(ledStatus ? "1" : "0", function(err, results) {});
});

/*
//
// Serial Port를 사용하는 예제
//
var SerialPort = require("serialport").SerialPort

// Arduino가 "COM7" 에 연결되었다고 가정한다.
var serialPort = new SerialPort("COM7", {baudrate: 9600, false});

serialPort.open(function () {
  console.log('접속되었습니다!');
  serialPort.on('data', function(data) {
    // Arduino에서 오는 데이터를 출력한다.
    console.log('data received: ' + data);
  });
  /*
  setInterval(function(){
    // LED가 ON/OFF 될꺼예요
    serialPort.write(ledStatus ? "1" : "0", function(err, results) {});
  }, 100);
  */
});
*/
//
// Johnny-five 모듈로 보드에 접속해 LED를 제어한다.

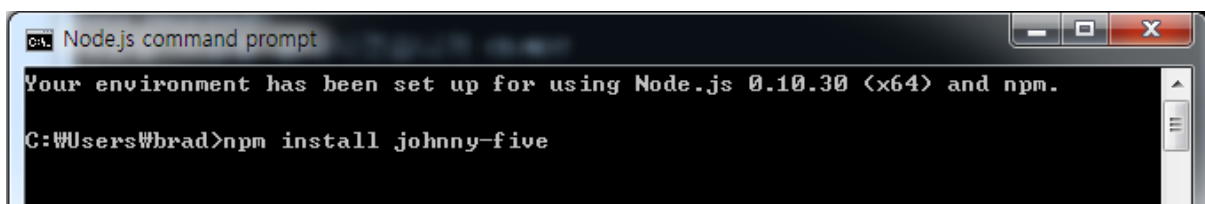
```

```
//
var five = require('johnny-five')
, board = new five.Board({port:"COM7"});

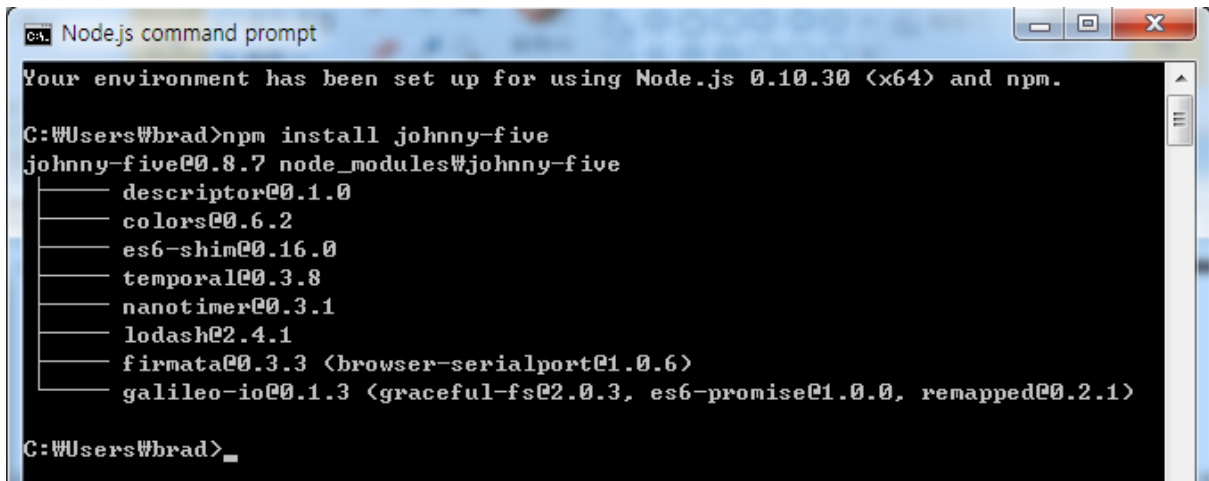
board.on("ready", function() {
  //
  // 13번 PIN을 OUTPUT으로
  //
  this.pinMode(13, 1);

  //
  // 0.1s마다 돌면서 ledStatus 값에 따라 LED를 ON/OFF 한다.
  //
  this.loop(100, function() {
    //
    this.digitalWrite( 13, ledStatus ? 1 : 0 );
  });
});
```

app.post() 일부분을 코멘트 처리해 준다. serialport 부분은 전체적으로 막아주고, johnny-five를 이용하는 부분을 추가해준다. 이 예제를 실행시키기 위해서는 johnny-five 모듈이 필요하다. 다음과 같이 [Node.js command prompt]를 새로 실행시켜 johnny-five 모듈을 설치해 준다.



다음과 같이 설치가 된다.

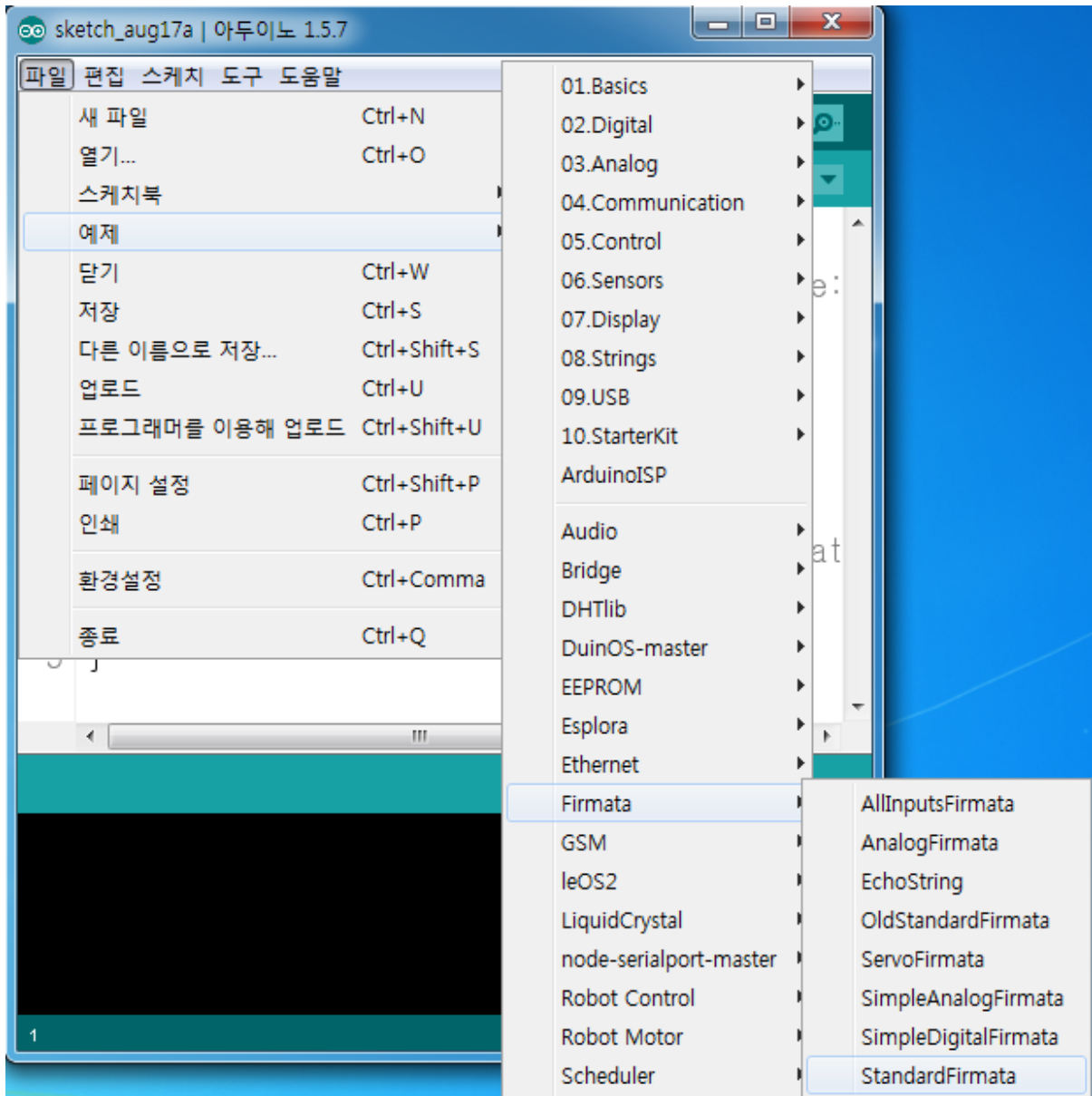


```
Node.js command prompt
Your environment has been set up for using Node.js 0.10.30 <x64> and npm.

C:\Users\Wbrad>npm install johnny-five
johnny-five@0.8.7 node_modules\johnny-five
├── descriptor@0.1.0
├── colors@0.6.2
├── es6-shim@0.16.0
├── temporal@0.3.8
├── nanotimer@0.3.1
├── lodash@2.4.1
├── firmata@0.3.3 <browser-serialport@1.0.6>
└── galileo-io@0.1.3 <graceful-fs@2.0.3, es6-promise@1.0.0, remapped@0.2.1>

C:\Users\Wbrad>_
```

이제 아두이노에 StandardFirmata 예제를 선택하여 업로드해준다.



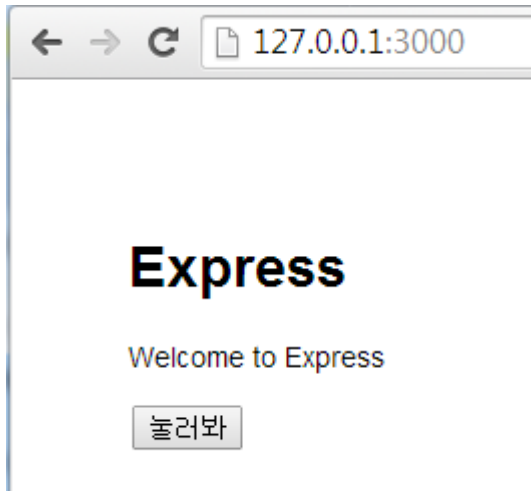
이제 다음과 같이 웹서버를 구동시킨다.

```
C:\Users\brad\tmp\foo>npm start

> application-name@0.0.1 start C:\Users\brad\tmp\foo
> node ./bin/www

1408235538085 Device(s) COM7
1408235541353 Connected COM7
1408235541356 Repl Initialized
>>
```

이제 웹브라우저로 다음과 같이 접속하여 버튼을 눌러본다.



LED가 켜졌다 꺼졌다 하는 것을 볼 수 있다.

다음은 웹서버 화면이다.

```
C:\Users\Wbrad\Wtmp\Wfoo>npm start

> application-name@0.0.1 start C:\Users\Wbrad\Wtmp\Wfoo
> node ./bin/www

1408235779456 Device(s) COM7
1408235782719 Connected COM7
1408235782721 Repl Initialized
>> POST / 200 25ms - 368b
true
GET /stylesheets/style.css 304 3ms
POST / 200 3ms - 368b
false
GET /stylesheets/style.css 304 1ms
POST / 200 2ms - 368b
true
GET /stylesheets/style.css 304 2ms
POST / 200 2ms - 368b
false
GET /stylesheets/style.css 304 1ms
```

NodeJS 말고 다른건 없나요?

<https://github.com/firmata/arduino>에 가시면 언어별 Firmata 클라이언트 라이브러리가 있으니 살펴보시면 도움이 될 것 같습니다.

이상 Node.js를 이용하여 아두이노를 제어해 봤다.