NODE JS

Версия NodeJS: **node-v**

Помощь: **.help**

Запуск NodeJS: **node**

После запуска **node** открывается консоль, в которой можно писать JS и он будет выполняться. Выйти из консоли можно с помощью команды **.exit** или сочетанием клавиш **CTRL+C.**

Пользоваться только консолью неудобно, поэтому лучше в ноде запускать файлы.

После создания файла index.js, мы можем запустить его в ноде, для этого в командной строке пишем: **node index** (расширение js можно не писать)

NPM

npm – менеджер пакетов

версия: **npm -v**

**Инициализация проекта**

Вызывается командой **npm init**

На этой стадии инициализируется проект, создается файл **package.json:**

{  
 "name": "nodejscourse",  
 "version": "1.0.0",  
 "description": "Node JS course for beginners",  
 "main": "index.js",  
 "scripts": {  
 "test": "echo \"Error: no test specified\" && exit 1"  
 },  
 "repository": {  
 "type": "git",  
 "url": "git+https://github.com/Smovg-by/nodejs\_minin.git"  
 },  
 "keywords": [  
 "node",  
 "nodejs"  
 ],  
 "author": "Uladzimir Smouzh <smovg1986@gmail.com>",  
 "license": "ISC",  
 "bugs": {  
 "url": "https://github.com/Smovg-by/nodejs\_minin/issues"  
 },  
 "homepage": "https://github.com/Smovg-by/nodejs\_minin#readme"  
}

В этом файле мы будем хранить список зависимостей для приложения.

**Установка пакетов (зависимостей)**

Вызывается командой **npm i <имя пакета>** (можно npm install <имя пакета>).

При установке первого пакета создается папка node modules

В этой папке ставятся папки зависимостей, в каждой есть тоже файл package.json. *Т.е. каждый пакет (зависимость) тянет за собой и другие зависимости.*

***Поле devDependencies***

Можно поставить пакеты, которые будут нужны только на стадии разработки. Установка таких пакетов происходит с помощью команды:

**npm i** <имя пакета> **-D**

**Импорт пакетов(модулей)**

Импорт пакетов делается с помощью слова **require()**. Свойства будут доступны по как методы переменной, например:

const chalk = require('chalk');

Можно создать свой модуль. Создадим файл data.js и сделаем его импортируемым модулем с помощью **module.exports()**:

const text = 'Hello Node JS from data.js';  
  
module.exports = text;

**Встроенные модули Node JS**

Их можно посмотреть на оф сайте в доках для текущей версии, например:

<https://nodejs.org/dist/latest-v14.x/docs/api/>

**- Модуль PATH**

Метод **basename** – выводим имя файла

*console*.log("Filename: ", path.basename(\_\_filename));

Метод **dirname** – выводит абсолютный путь к директории и ее имя

*console*.log("Directory name: ", path.dirname(\_\_filename));

Метод **extname** – выводит расширение файла

*console*.log("File Extension", path.extname(\_\_filename));

Метод **parse** – универсальный метод, выводит объект с данными

*console*.log ('Parse: ', path.parse(\_\_filename));

Parse: {

root: 'C:\\',

dir: 'C:\\Users\\user\\Desktop\\nodeJSCourse\\demo',

base: 'path.js',

ext: '.js',

name: 'path'

}

Метод **join** – создает путь к файлу

*console*.log(path.join(\_\_dirname, 'server', 'index.html'));

**вывод: C:\Users\user\Desktop\nodeJSCourse\demo\server\index.html**

**- Модуль FS (file system)**

Метод **mkdir** – создает директорию.

**!!!** Node JS является однопоточным, поэтому предлагает создание папок/файлов СИНХРОННЫМ методом и АСИНХРОННЫМ методом. В данном случае **mkdir** является *асинхронным методом*, поэтому мы должны обязательно использовать колл-бек. Также можно воспользоваться СИНХРОННЫМ методом **mkdirSync, НО ЭТО БУДЕТ БЛОКИРОВАТЬ ПОТОК.**

Чтобы воспользоваться методом, нам понадобится библиотека path, т.к. **первым параметром передается путь**, **вторым имя папки**, **третьим идет колл-бек (если используется асинхронный метод).**

const fs = require('fs'); // fs - filesystem  
const path = require('path');  
  
fs.mkdir(path.join(\_\_dirname, 'test'), (*err*) => {  
 if (*err*) {  
 throw *err*;  
 }  
 *console*.log('Directory created')  
});

Метод **writeFile** – создает файл

Может перетирать существующие файлы!!!

const filePath = path.join(\_\_dirname, 'test', 'text.txt');  
  
fs.writeFile(filePath, 'Hello NODE JS again', (*err*)=>{  
 if (*err*) {  
 throw *err*;  
 }  
 *console*.log('File was created')  
});

Метод **appendFile** – добавляет содержимое к существующему файлу

const filePath = path.join(\_\_dirname, 'test', 'text.txt');  
  
fs.appendFile(filePath, '\n Hello NODE JS again and again', (*err*)=>{  
 if (*err*) {  
 throw *err*;  
 }  
 *console*.log('File was updated')  
});

Метод **readFile** – чтение содержимого файла. Возвращает ОБЪЕКТ БУФЕРА

const filePath = path.join(\_\_dirname, 'test', 'text.txt');  
  
fs.readFile(filePath, (*err*, *content*) => {  
 if (*err*) {  
 throw *err* }  
 *console*.log(*content*)  
});

Есть несколько способов преобразовать буферные данные в строку.

1. **Buffer.from(content)**

const filePath = path.join(\_\_dirname, 'test', 'text.txt');  
  
fs.readFile(filePath, (*err*, *content*) => {  
 if (*err*) {  
 throw *err* }  
 const data = Buffer.from(*content*);  
 *console*.log('Content', data.toString())  
});

1. **Использование кодировки (предпочтительнее)**

const filePath = path.join(\_\_dirname, 'test', 'text.txt');  
  
fs.*readFile*(filePath, 'utf-8',(*err*, *content*) => {  
 if (*err*) {  
 throw *err* }  
 console.log('Content', *content*)  
});