



MODULE 2: NODEJS SYNCHRONOUS FILE SYSTEMS APIS

IT 207 – IT Programming

LECTURE OUTLINE

- ❖ Overview of the file System.
- ❖ Accessing File system in Nodejs
- ❖ Nodejs Synchronous directory and file APIs
- ❖ Node Data types
- ❖ Command line arguments in Node

OVERVIEW OF THE FILESYSTEM



WHAT IS FILESYSTEM?

- ❖ The filesystem is a major component of any operating system
- ❖ The file system is responsible for **persistently** storing, managing and updating data on the storage device in question.
- ❖ Different OS use different file systems
 - ❖ Windows: FAT, NTFS, exFAT
 - ❖ MAC: macOS HFS, APFS, HFS+
 - ❖ Linux: EXT2/3/4, XFS, JFS, Btrfs
- ❖ The file system exports two abstractions
 - ❖ Files
 - ❖ Directories

FILES

- ❖ Files are the smallest unit of persistent data storage
- ❖ A file is a linear array of bytes, each of which you can be read or written.
- ❖ A file is identified by a
 - ❖ A textual name that is composed of two parts separated by a dot '.'
 - ❖ Base name: a unique name identifying the file within its directory
 - ❖ extension that indicates the file type
 - ❖ Low-level name usually a number

FILE PROPERTIES

- ❖ A file system keeps a number of attributes for each file
 - ❖ File size measured in bytes
 - ❖ File creation and access dates
 - ❖ The default application to open the file with
 - ❖ Access permissions that determine who can read, write and execute the file

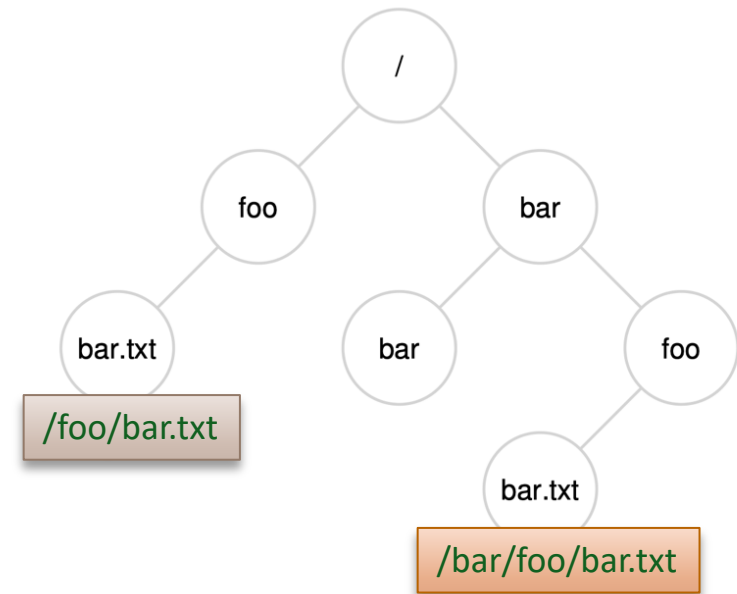
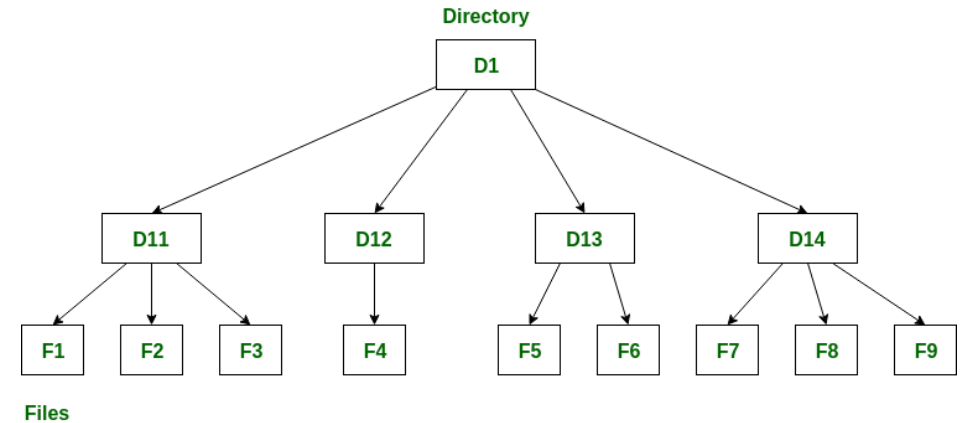
```
zaira@Zaira:~/freeCodeCamp$ ls -l
total 3856
-rw-r--r--  1 zaira zaira   89 Apr  5 20:46 CODE_OF_CONDUCT.md
-rw-r--r--  1 zaira zaira  210 Apr  5 20:46 CONTRIBUTING.md
-rw-r--r--  1 zaira zaira 1513 Apr  5 20:46 LICENSE.md
-rw-r--r--  1 zaira zaira 19933 Apr  5 20:46 README.md
drwxr-xr-x  4 zaira zaira 4096 Apr  6 22:45 api-server
-rw-r--r--  1 zaira zaira   67 Apr  5 20:46 babel.config.js
drwxr-xr-x 10 zaira zaira 4096 Apr  6 22:55 client
drwxr-xr-x  5 zaira zaira 4096 Apr  6 22:54 config
```

MODE OWNER GROUP SIZE MODIFICATION DATE FILE/FOLDER NAME

<https://www.freecodecamp.org/news/linux-chmod-chown-change-file-permissions/>

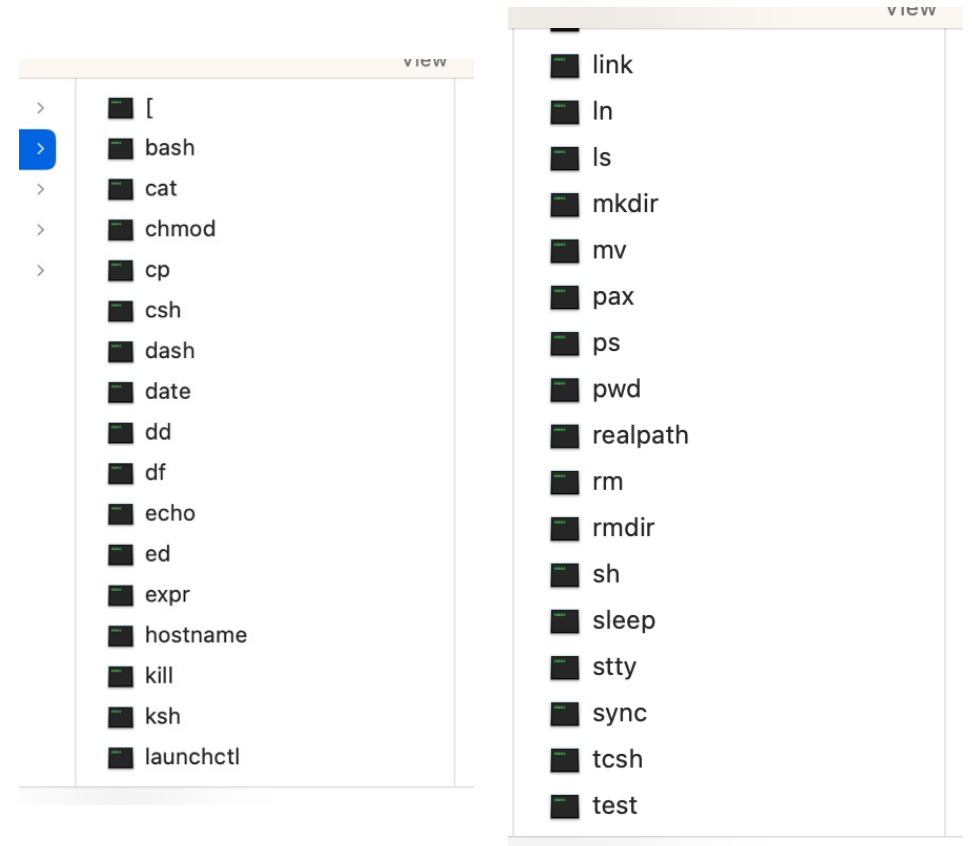
DIRECTORIES

- ❖ A directory is a **container** for files
- ❖ Directories are organized in **hierarchal** structure such as a tree.
- ❖ Like files a directory is identified by a **textual** name and a low-level name
- ❖ The content of the directory is a list of user-readable name and the low-level name of the files that it contains.
- ❖ A **path** is the sequence of **directory names** that starts from the root directory and specifies subsequent directories until the desired file or path is reached
 - ❖ The path uses a special character to separate the Directories names
- ❖ **./** and **../** are two special directory names that exist in every directory
 - ❖ **./** refers to the **current** directory
 - ❖ **../** refers to the **parent** directory



COMMAND LINE FILE & DIRECTORY COMMANDS

- ❖ The OS provides several file and directory commands that can be executed at the **command prompt**.
- ❖ `cat`: print file content
- ❖ `cp`: copy file
- ❖ `mkdir`: make directory
- ❖ `ls`: list files and directories
- ❖ `rm`: remove file
- ❖ `rmdir`: remove directory
- ❖ `pwd`: print working directory
- ❖ `mv`: move file
- ❖ `touch`: create, change and modify timestamps of a file



Where is `cd`?

NODEJS
FILESYSTEM



NODEJS FILESYSTEM SUPPORT

❖ Built-in Modules

- ❖ `path`: provides utilities for working with file and directory paths
- ❖ `fs`: enables interacting with the file system in a way modeled on standard POSIX functions.

❖ Environment variables

- ❖ `__filename`: holds the absolute file name of the current module.
- ❖ `__dirname`: holds the directory name of the current module.
- ❖ The `__dirname` can be also obtained using the `path.dirname()` with `__filename` as the argument.

```
console.log(__filename);  
// Prints: /Users/mjr/example.js  
  
console.log(__dirname);  
// Prints: /Users/mjr
```

NODEJS FS MODULE

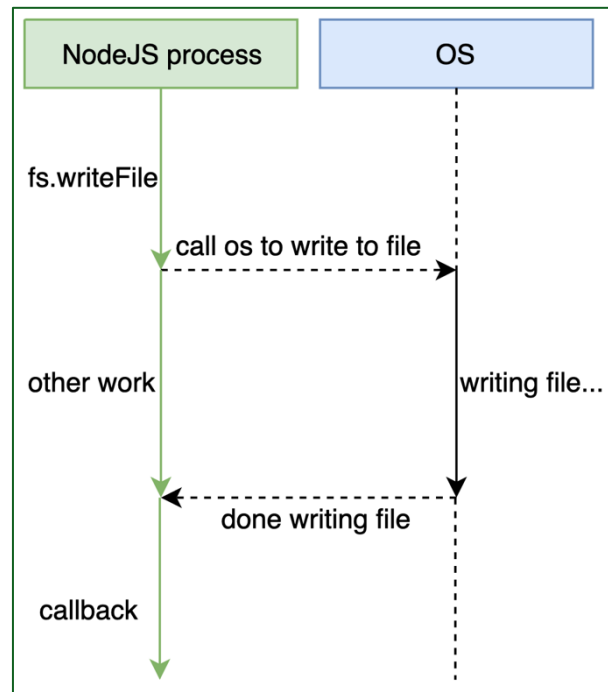
❖ The Nodejs fs module provides 3 categories of APIs to access files:

❖ Synchronous APIs

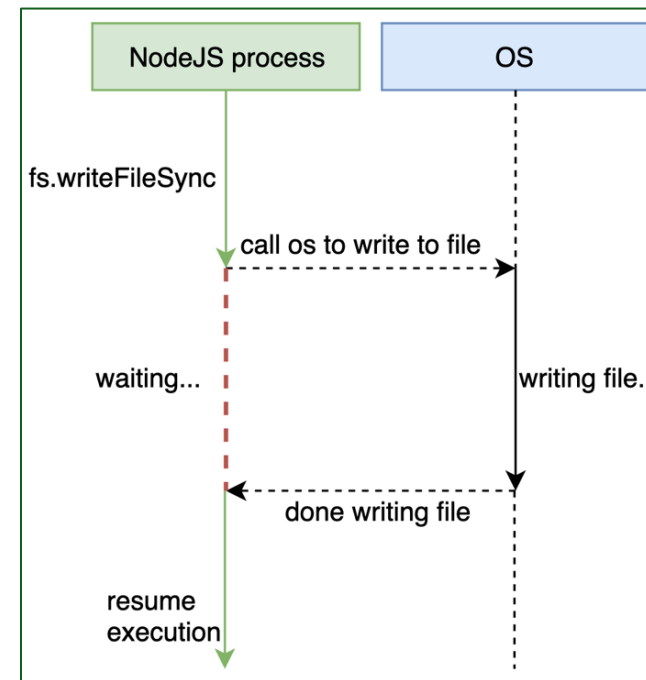
❖ Callback APIs

❖ Promise APIs

Asynchronous APIs



Asynchronous APIs



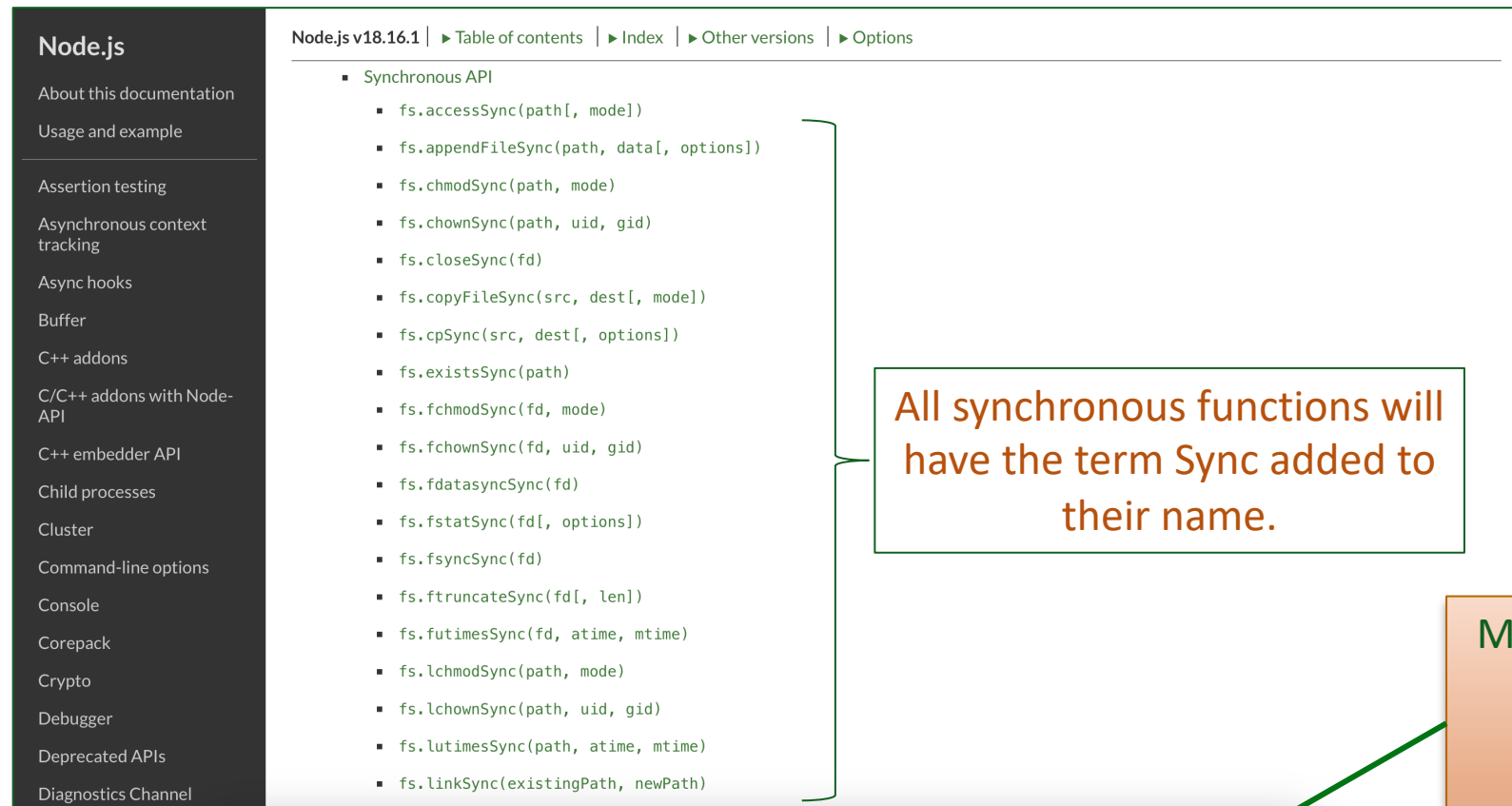
Synchronous APIs

FILE SYSTEM
SYNCHRONOUS
APIS



NODEJS DOCUMENTATION

❖ Nodejs Synchronous APIs are listed under the File system Node documentation



The screenshot shows the Node.js v18.16.1 documentation for the File System (fs) module. The page title is "Node.js v18.16.1" with navigation links for "Table of contents", "Index", "Other versions", and "Options". A sidebar on the left lists various Node.js topics. The main content area is titled "Synchronous API" and lists 20 functions, each with its signature. A green bracket on the right side of the list groups all these functions together, pointing to an orange callout box.

Node.js v18.16.1 | ▶ Table of contents | ▶ Index | ▶ Other versions | ▶ Options

- Synchronous API
 - `fs.accessSync(path[, mode])`
 - `fs.appendFileSync(path, data[, options])`
 - `fs.chmodSync(path, mode)`
 - `fs.chownSync(path, uid, gid)`
 - `fs.closeSync(fd)`
 - `fs.copyFileSync(src, dest[, mode])`
 - `fs.cpSync(src, dest[, options])`
 - `fs.existsSync(path)`
 - `fs.fchmodSync(fd, mode)`
 - `fs.fchownSync(fd, uid, gid)`
 - `fs.fdatasyncSync(fd)`
 - `fs.fstatSync(fd[, options])`
 - `fs.fsyncSync(fd)`
 - `fs.ftruncateSync(fd[, len])`
 - `fs.futimesSync(fd, atime, mtime)`
 - `fs.lchmodSync(path, mode)`
 - `fs.lchownSync(path, uid, gid)`
 - `fs.lutimesSync(path, atime, mtime)`
 - `fs.linkSync(existingPath, newPath)`

All synchronous functions will have the term Sync added to their name.

Make sure to check the documentation corresponding your Nodejs version

<https://nodejs.org/docs/latest-v18.x/api/fs.html>

DIR FUNCTIONS

- ❖ Node provides support to several directory command.
 - ❖ Always check the function details and arguments on the documentation page
- ❖ Example:
 - ❖ Create a directory: `fs.mkdirSync(path[, options])`
 - ❖ Read the contents of a directory: `fs.readdirSync(path[, options])`
 - ❖ Remove a directory: `fs.rmdirSync (path[, options])`

```
//dir.js

const fs = require('fs');
//Get the file list of the current directory
let list = fs.readdirSync(__dirname);
console.log("The list of files in the directory:\n"+list);

//make a new directory called new
let newdir = __dirname + '/new';
fs.mkdirSync(newdir);

//Get the file list of the current directory
list = fs.readdirSync(__dirname,"utf-8");
console.log("The list of files in the directory:\n"+list);
```

```
The list of files in the directory:
dir.js
The list of files in the directory:
dir.js,new
```

FILE FUNCTIONS

- ❖ Node provides support to many file functions.
 - ❖ Always check the function details and arguments on the documentation page
- ❖ Examples (*Not an exhaustive list*)
 - ❖ Reading from a File
 - ❖ `readFileSync(path[, options])`
 - ❖ `readSync(fd, buffer, offset, length[, position])`
 - ❖ `readSync(fd, buffer[, options])`
 - ❖ Reading the status of a file
 - ❖ `statSync(path[, options])`
 - ❖ Writing to a file
 - ❖ `writeFileSync(file, data[, options])`
 - ❖ `writeSync(fd, buffer, offset[, length[, position]])`
 - ❖ `writeSync(fd, buffer[, options])`
 - ❖ `writeSync(fd, string[, position[, encoding]])`
 - ❖ `appendFileSync(path, data[, options])`

READING FROM A FILE

❖ fs.readFileSync:

- ❖ Reads the full content of a file into memory
- ❖ Suitable for small files. Not recommended for large files.

fs.readFileSync(path[, options])

► History

- `path` `<string>` | `<Buffer>` | `<URL>` | `<integer>` filename or file descriptor
- `options` `<Object>` | `<string>`
 - `encoding` `<string>` | `<null>` **Default:** `null`
 - `flag` `<string>` See [support of file system flags](#). **Default:** `'r'`.
- Returns: `<string>` | `<Buffer>`

Returns the contents of the `path`.

READING FROM A FILE

- ❖ fs.readFileSync:
 - ❖ Reads the full content of a file into memory
 - ❖ Suitable for small files. Not recommended for large files

```
const fs = require('fs');
//Read the content of the file as one big chunk into a buffer
//1.Path=>String - Default => sequence of bytes
let content = fs.readFileSync("./Source.txt");
console.log(content);
//2. Path=> String - options=> object{encoding:"utf-8"}
content = fs.readFileSync("./Source.txt",{encoding:"utf-8"});
console.log(content);
//3. Path=> String - options=> object{encoding:"utf-8",flag:"r"}
content = fs.readFileSync("./Source.txt",{encoding:"utf-8",flag:"r"});
console.log(content);
//4.Path => file descriptor - options=>object{encoding:"utf-8"}
let fd = fs.openSync("./Source.txt", "r");
console.log('File descriptor = '+fd);
content = fs.readFileSync(fd,{encoding:"utf-8",flag:"r"});
console.log(content);
fs.closeSync(fd);
```

WRITING TO A FILE

❖ Writing file

- ❖ Writes different formats to a file.
- ❖ Suitable of writing a small chunk once. Not recommended for writing large chunks or writing continuously to the file.

```
fs.writeFileSync(file, data[, options])
```

► History

- `file` `<string>` | `<Buffer>` | `<URL>` | `<integer>` filename or file descriptor
- `data` `<string>` | `<Buffer>` | `<TypedArray>` | `<DataView>` | `<Object>`
- `options` `<Object>` | `<string>`
 - `encoding` `<string>` | `<null>` **Default:** `'utf8'`
 - `mode` `<integer>` **Default:** `0o666`
 - `flag` `<string>` See [support of file system flags](#). **Default:** `'w'`.

Returns `undefined`.

WRITING TO A FILE

❖ Writing file

- ❖ Writes different formats to a file.
- ❖ Suitable of writing a small chunk once. Not recommended for writing large chunks or writing continuously to the file.

`fs.writeFileSync(file, data[, options])`

► History

- `file` `<string>` | `<Buffer>` | `<URL>` | `<integer>` filename or file descriptor
- `data` `<string>` | `<Buffer>` | `<TypedArray>` | `<DataView>` | `<Object>`
- `options` `<Object>` | `<string>`
 - `encoding` `<string>` | `<null>` **Default:** `'utf8'`
 - `mode` `<integer>` **Default:** `0o666`
 - `flag` `<string>` See [support of file system flags](#) . **Default:** `'w'` .

File access permission:
6 => read and write permission

Returns `undefined` .

WRITING TO A FILE

- ❖ Writing file
 - ❖ Writes different formats to a file.
 - ❖ Suitable of writing a small chunk once. Not recommended for writing large chunks or writing continuously to the file.

```
//This code will read a file and copy its content into another file
const fs = require('fs');
/** 1. Read in as a sequence of bytes into a buffer from 'simple.txt' and */
//Path => string - Options=> sequence of bytes
let data = fs.readFileSync('simple.txt');
console.log(data);
console.log(data.toString()); //convert the byte to equivalent string using utf-8
/** 2. write the buffer back 'simpleCopy.txt'*/
//Default: use utf-8 encoding - same as data.toString() or data.String('utf-8')
fs.writeFileSync("./simpleCopy.txt",data);
//use hexa values
fs.writeFileSync("./simpleCopy1.txt",data.toString('hex'));
//use option object - utf-8 will always be used regardless of the encoding value
fs.writeFileSync("./simpleCopy2.txt",data,{encoding:'hex',mode:0o666,flag:"w"});
/** 3.Append a string to "./simpleCopy.txt" */
fs.appendFileSync("./simpleCopy.txt",'This will be at the end of the file');
```

WRITING TO A FILE

❖ Writing file

- ❖ Writes different formats to a file.
- ❖ Suitable of writing a small chunk once. Not recommended for writing large chunks or writing continuously to the file.

```
//This code will read a file and copy its content into another file
const fs = require('fs');
/** 1. Read in as a sequence of bytes into a buffer from 'simple.txt' and */
//Path => string - Options=> sequence of bytes
let data = fs.readFileSync('simple.txt');
console.log(data);
console.log(data.toString()); // this is a simple text file
/** 2. write the buffer back 'simpleCopy.txt' containing three lines of text
//Default: use utf-8 encoding - This is the last line.
fs.writeFileSync('./simpleCopy.txt', data);
//use hexa values
fs.writeFileSync('./simpleCopy1.txt', data.toString('hex'));
//use option object - utf-8 will always be used regardless of the encoding value
fs.writeFileSync('./simpleCopy2.txt', data, {encoding: 'hex', mode: 0o666, flag: "w"});
/** 3. Append a string to './simpleCopy.txt' */
fs.appendFileSync('./simpleCopy.txt', 'This will be at the end of the file');
```

WRITING TO A FILE

❖ Writing file

- ❖ Writes different formats to a file.
- ❖ Suitable of writing a small chunk once. Not recommended for writing large chunks or writing continuously to the file.

```
//This code will read a file and copy its content into another file
const fs = require('fs');
/** 1. Read in as a sequence of bytes into a buffer from 'simple.txt' and */
//Path => string - Options=> sequence of bytes
let data = fs.readFileSync('simple.txt');
console.log(data);
console.log(data.toString()); // this is a simple text file
/** 2. write the buffer back 'simpleCopy.txt' containing three lines of text
//Default: use utf-8 encoding - This is the last line.
fs.writeFileSync('./simpleCopy.txt', data);
//use hexa values
fs.writeFileSync('./simpleCopy1.txt', data.toString('hex'));
//use option object - utf-8 will always be used regardless of the encoding value
fs.writeFileSync('./simpleCopy2.txt', data, {encoding: 'hex', mode: 0o666, flag: "w"});
/** 3. Append a string to './simpleCopy.txt' */
fs.appendFileSync('./simpleCopy.txt', 'This will be at the end of the file');
```

Append adds to the end of the file

NODE DATA TYPES



NODE OBJECTS

- ❖ Same as Objects in JS
- ❖ A Node object is a container for key: value pairs for collection and/or complex entities.
- ❖ The key : value pairs are called object properties

```
//declaring empty object
const obj1 = {};
console.log(obj1); // {}
//add property to object
obj1.key1 = 'value1';
console.log(obj1); //{ key1: 'value1' }
//declare another object with properties
const obj2 = {
  text:"Hello world",
  num: 3,
  boolean: false,
  bigNum: 2.3e+10
}
```

```
//add function as a property
obj2.print = function(){
  console.log(obj2);
}
//call obj2 function
obj2.print(); /* {
  text: 'Hello world',
  num: 3,
  boolean: false,
  bigNum: 23000000000,
  print: [Function (anonymous)]
}*/
```


NODE BUFFER

- ❖ Special type of objects that handles binary data
 - ❖ Buffer objects are used to represent a fixed-length sequence of bytes.
<https://nodejs.org/api/buffer.html#buffer>
- ❖ Buffer class is available within the global scope
- ❖ Node provides several functions that work on buffers as (not an exhaustive list)
 - ❖ `alloc()`: Creates a Buffer object of the specified length
 - ❖ `toString()`: Returns a string version of a Buffer object
 - ❖ `write()`: Writes a specified string to a Buffer object
 - ❖ `length`: Returns the length of a Buffer object, in bytes

```
let buff = Buffer.alloc(8);  
console.log(buff);  
buff.write('abc');  
console.log(buff);  
console.log(buff.toString());
```

```
<Buffer 00 00 00 00 00 00 00 00>  
<Buffer 61 62 63 00 00 00 00 00>  
abc
```

COMMAND LINE ARGUMENTS



COMMAND LINE ARGUMENTS IN NODE - PROCESS.ARGV

- ❖ Nodejs provides the *process* object which has information about, and control over, the current Nodejs process.
- ❖ One of the properties of the *process* object is the argument vector *argv*.
- ❖ `process.argv` returns an **array** containing the **command-line arguments** passed when the Node.js process was launched.

```
//process-arg.js is a script that prints process.argv
//Execute the Script as node process-args.js one two=three four

process.argv.forEach((val, index)=>{
  console.log(` ${index}:${val}`); //Template String
});
```

Launch the script as:

```
node process-args.js one two=three four
```

COMMAND LINE ARGUMENTS IN NODE

- ❖ Nodejs provides the *process* object which has information about, and control over, the current Nodejs process.
- ❖ One of the properties of the *process* object is the argument vector *argv*.
- ❖ `process.argv` returns an **array** containing the **command-line arguments** passed when the Node.js process was launched.

```
//process-arg.js is a script that prints process.argv
//Execute the Script as node process-args.js one two=three four

process.argv.forEach((val, index)=>{
  console.log(` ${index}:${val} `); //Template String
});
```

```
CodeEx % node process-args.js one two=three four
0:/usr/local/bin/node
1:/Users/hmhassan/Module 2/CodeEx/process-args.js
2:one
3:two=three
4:four
```

SUMMARY 1

- ❖ The filesystem is a major component of any operating system that handles the storage, access, management and update of persistent data
- ❖ The filesystem files and directories as two abstractions for storing persistent data
- ❖ Files store data as a linear sequence of bytes
- ❖ Files are identified by a name that is composed of an extension indicating the type of the file
- ❖ Files have several fields associated with it indicating its size, date of creation and access, access rights
- ❖ Directories are containers files stored using a hierarchical structure
- ❖ Node provides different APIs for handling files and directories
- ❖ The Synchronous API handles all file operations in Node's event loop
- ❖ the buffer module in node defines the buffer class that handles binary data

SUMMARY 2

- ❖ The process object is a global object in Nodejs that has information about, and control over, the current Nodejs process.
- ❖ `process.argv` holds the command line arguments passed to the Nodejs process when it is launched
- ❖ `forEach()` is a JS method that calls an anonymous function for each element in the array