Node.js Cheat Sheet

Modules · HTTP · Streams · Events · Async

worksheetdownload.com

{ }

SETUP & MODULES

```
# CommonJS (CJS):
const fs = require('fs')
module.exports = { fn }
# ESM (package.json: "type":"module"):
import fs from 'fs'
export const fn = () => {}
node --version  /  nvm use 20
```

NPM BASICS

```
npm init -y
npm install express  (saves to
dependencies)
npm install -D jest  (saves to
devDependencies)
npm run start  /  npm test
npm ci  (clean install from
package-lock.json)
npx create-next-app  (run without
install)
```

HTTP SERVER

```
import http from 'http'
const server = http.createServer((req,
res) => {
res.writeHead(200, {'Content-Type':
'application/json'})
res.end(JSON.stringify({ ok: true }))
})
server.listen(3000, () =>
console.log('listening'))
```

Most projects use Express/Fastify on top of http

FILE SYSTEM

```
import { promises as fs } from 'fs'
const data = await
fs.readFile('./file.txt', 'utf8')
await fs.writeFile('./out.txt', data)
await fs.mkdir('./dir', { recursive:
true })
const files = await fs.readdir('./dir')
await fs.rename(old, new) /
fs.unlink(path)
```

Always prefer async (promises) over sync methods

EVENTS (EVENTEMITTER)

```
import { EventEmitter } from 'events'
const emitter = new EventEmitter()
emitter.on('data', (payload) => { ... })
emitter.once('end', () => cleanup())
emitter.emit('data', { id: 1 })
emitter.removeListener('data', handler)
emitter.setMaxListeners(20) // avoid
memory leak warn
```

STREAMS

```
import { createReadStream,
createWriteStream } from 'fs'
createReadStream('input.txt')
.pipe(zlib.createGzip())
.pipe(createWriteStream('output.gz'))
// Async iteration:
for await (const chunk of
readableStream) {
process(chunk)
}
```

ASYNC PATTERNS

```
// Callback (legacy):
fs.readFile('a.txt', (err, data) => {})
// Promises:
const data = await
fs.promises.readFile('a.txt')
// Parallel:
const [a, b] = await
Promise.all([fetch1(), fetch2()])
// promisify legacy callbacks:
const readFile =
util.promisify(fs.readFile)
```

BUFFERS

```
const buf = Buffer.from('hello', 'utf8')
buf.toString('base64')
Buffer.alloc(256)  // zero-filled buffer
buf.length  /  buf.slice(0, 4)
Buffer.concat([buf1, buf2])
```

Use for binary data, crypto, networking

PROCESS & ENV VARS

```
process.env.PORT ?? 3000
process.env.NODE_ENV
process.argv  // command-line args
process.exit(0) / process.exit(1)
process.cwd()  // current working
directory
process.on('uncaughtException', handler)
process.on('unhandledRejection',
handler)
```

CHILD PROCESSES

```
import { exec, spawn } from
'child_process'
const { stdout } = await exec('ls -la')
const child = spawn('node',
['script.js'])
child.stdout.on('data', data =>
process.stdout.write(data))
child.on('exit', code =>
console.log(code))
```

spawn = streaming; exec = buffered; fork = for Node scripts

COMMON GLOBALS

```
__dirname  __filename   // CJS only
import.meta.url        // ESM
globalThis  global     // global object
setTimeout setInterval clearTimeout
clearInterval
setImmediate() / process.nextTick()
console.log warn error time timeEnd
JSON.stringify / JSON.parse
```

COMMON GOTCHAS

Single-threaded blocking I/O blocks ALL requests
Unhandled promise rejections crash process (Node 15)
process.nextTick runs before I/O events can starve I/O
require() caches modules mutations persist
ESM vs CJS interop: .mjs vs .cjs extensions or package.json type