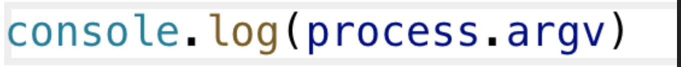
Node.JS Global

Command Line Tools. Debugging. Error Handling.

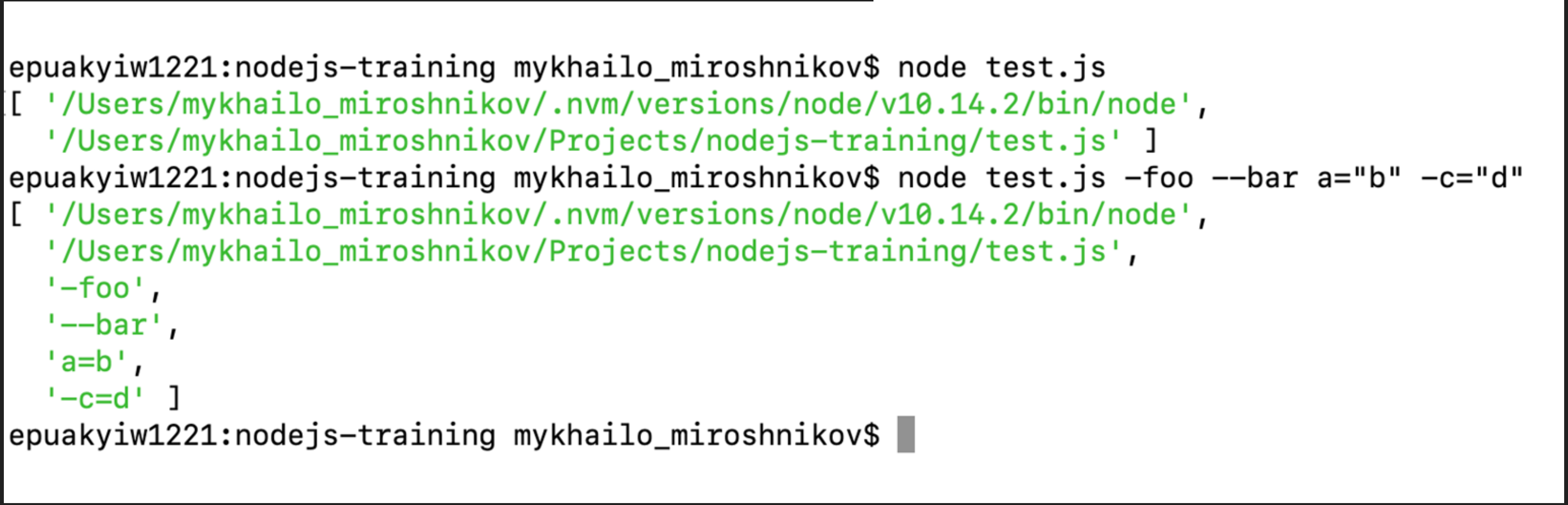
CLI Tools in Node.JS: `process.argv`

We need them to create custom command-line interfaces.

Main point of integration - **process.argv**

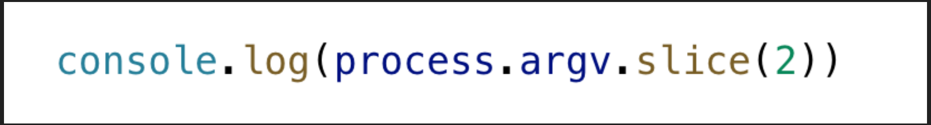


```js
console.log(process.argv)
```

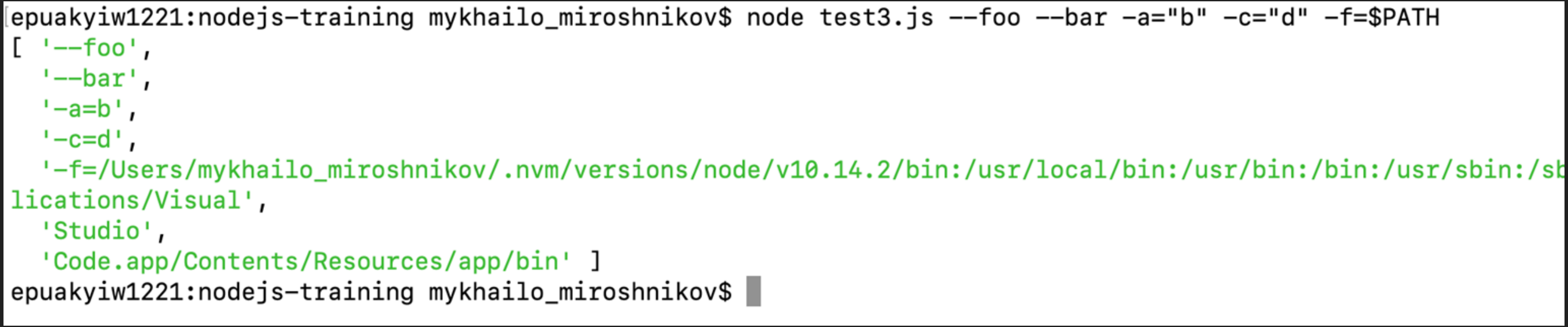


```
epuakyiw1221:nodejs-training mykhailo_miroshnikov$ node test.js
[ '/Users/mykhailo_miroshnikov/.nvm/versions/node/v10.14.2/bin/node',
  '/Users/mykhailo_miroshnikov/Projects/nodejs-training/test.js' ]
epuakyiw1221:nodejs-training mykhailo_miroshnikov$ node test.js -foo --bar a="b" -c="d"
[ '/Users/mykhailo_miroshnikov/.nvm/versions/node/v10.14.2/bin/node',
  '/Users/mykhailo_miroshnikov/Projects/nodejs-training/test.js',
  '-foo',
  '--bar',
  'a=b',
  '-c=d' ]
epuakyiw1221:nodejs-training mykhailo_miroshnikov$ 
```

`process.argv`: slice it!



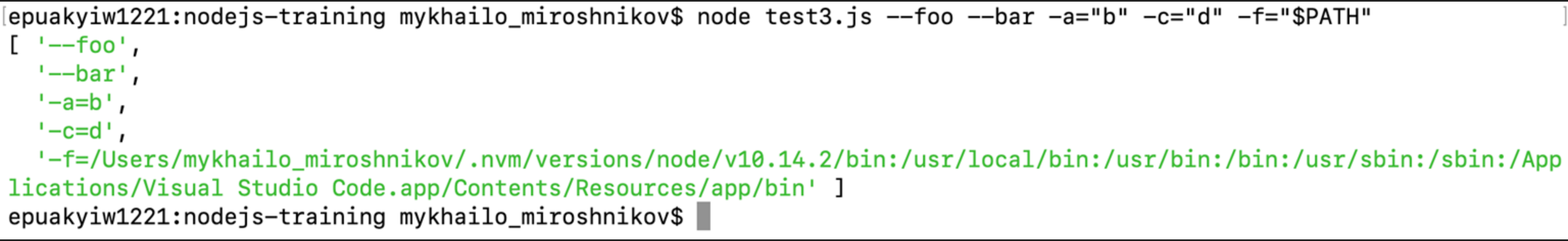
```
console.log(process.argv.slice(2))
```



```
epuakyiw1221:nodejs-training mykhailo_miroshnikov$ node test3.js --foo --bar -a="b" -c="d" -f=$PATH
[ '--foo',
  '--bar',
  '-a=b',
  '-c=d',
  '-f=/Users/mykhailo_miroshnikov/.nvm/versions/node/v10.14.2/bin:/usr/local/bin:/usr/bin:/bin:/usr/sbin:/sb
lications/Visual',
  'Studio',
  'Code.app/Contents/Resources/app/bin' ]
epuakyiw1221:nodejs-training mykhailo_miroshnikov$ 
```

`process.argv`: wrap it!

Wrap variables in quotes to avoid corruption:



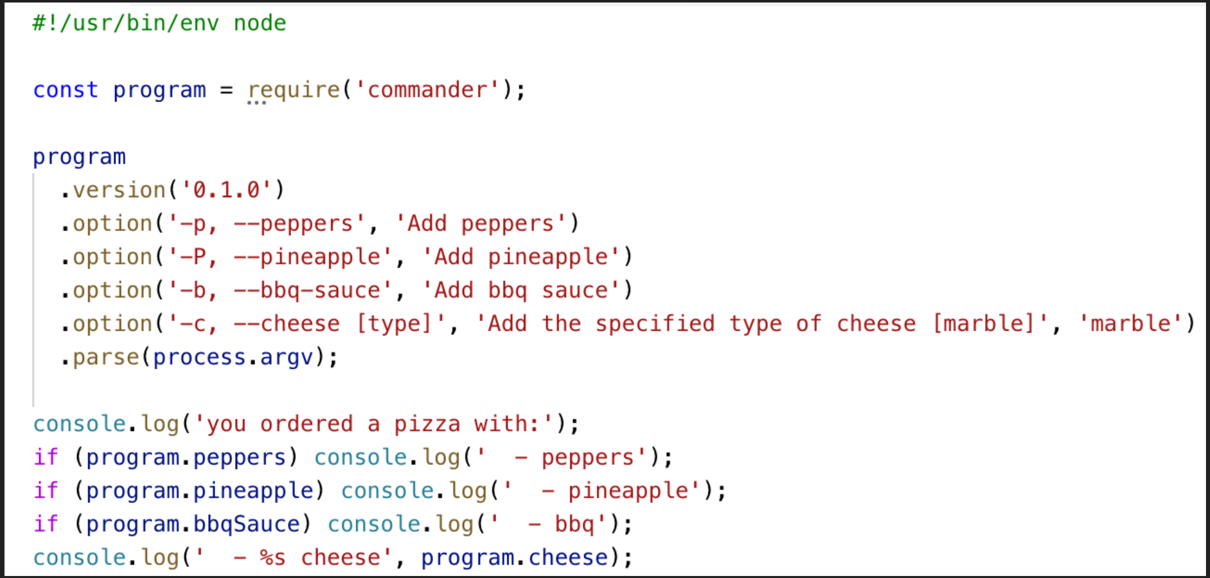
```
epuakyiw1221:nodejs-training mykhailo_miroshnikov$ node test3.js --foo --bar -a="b" -c="d" -f="$PATH"
[ '--foo',
  '--bar',
  '-a=b',
  '-c=d',
  '-f=/Users/mykhailo_miroshnikov/.nvm/versions/node/v10.14.2/bin:/usr/local/bin:/usr/bin:/bin:/usr/sbin:/sbin:/App
lications/Visual Studio Code.app/Contents/Resources/app/bin' ]
epuakyiw1221:nodejs-training mykhailo_miroshnikov$ 
```

NPM CLI Packages

commander

13K+ stars on GitHub

- aliases
- negating
- camel-casing
- descriptions
- etc.



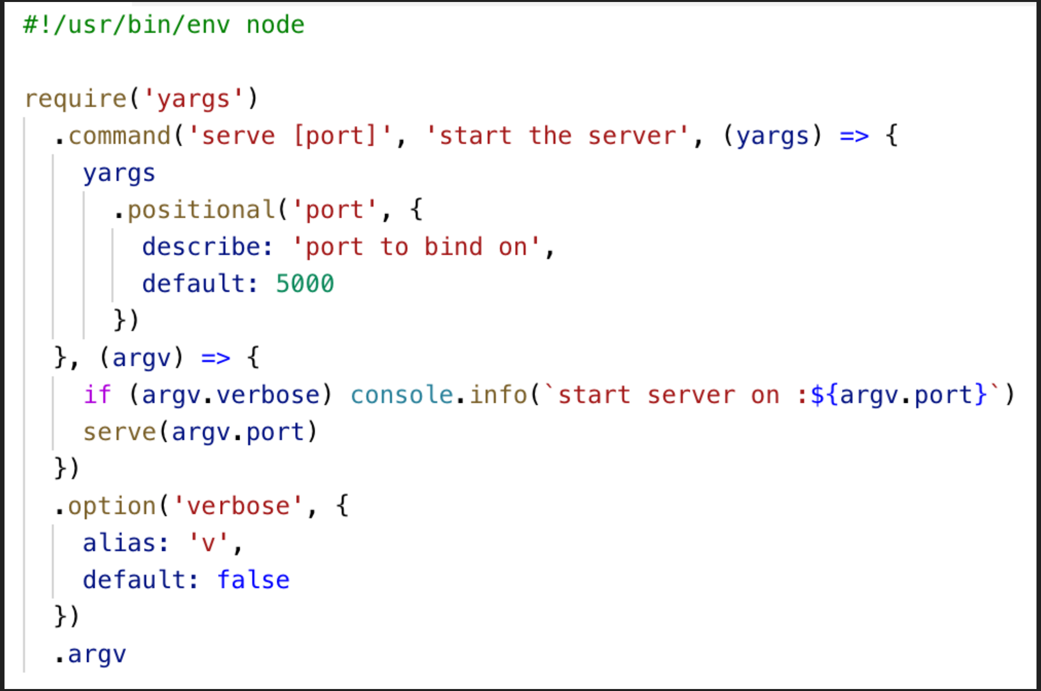
```
#!/usr/bin/env node

const program = require('commander');

program
  .version('0.1.0')
  .option('-p, --peppers', 'Add peppers')
  .option('-P, --pineapple', 'Add pineapple')
  .option('-b, --bbq-sauce', 'Add bbq sauce')
  .option('-c, --cheese [type]', 'Add the specified type of cheese [marble]', 'marble')
  .parse(process.argv);

console.log('you ordered a pizza with:');
if (program.peppers) console.log('  - peppers');
if (program.pineapple) console.log('  - pineapple');
if (program.bbqSauce) console.log('  - bbq');
console.log('  - %s cheese', program.cheese);
```

NPM CLI Packages

yargs

5.5K+ stars on GitHub

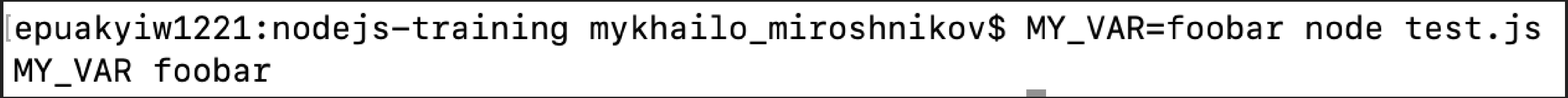
- aliases
- descriptions
- etc.



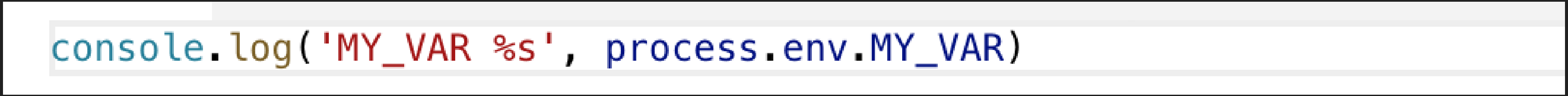
```
#!/usr/bin/env node

require('yargs')
  .command('serve [port]', 'start the server', (yargs) => {
    yargs
      .positional('port', {
        describe: 'port to bind on',
        default: 5000
      })
  }, (argv) => {
    if (argv.verbose) console.info(`start server on :${argv.port}`)
    serve(argv.port)
  })
  .option('verbose', {
    alias: 'v',
    default: false
  })
  .argv
```

ENV Variables

Rule #0: **Don’t modify environment variables**



```
epuakyiw1221:nodejs-training mykhailo_miroshnikov$ MY_VAR=foobar node test.js
MY_VAR foobar
```

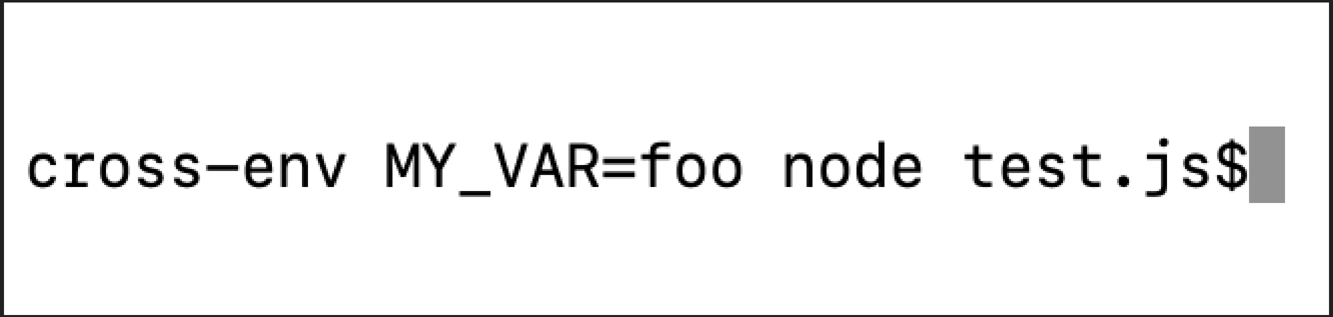


```
console.log('MY_VAR %s', process.env.MY_VAR)
```

Cross-platform ENV variables

Use cross-env package to be able to reuse your CLI scripts on different platforms.

Only UNIX terminals support raw env variables.



```
cross-env MY_VAR=foo node test.js$
```

Node.JS REPL - Read-Evaluate-Print Loop

Allows to evaluate JavaScript on-the-fly **(sometimes useful)**

Core modules don’t need to be **`require()`**’ed **(useful)**

Supports autocomplete **(useful)**

Supports different commands **(mostly, useless)**

`require('repl')`

Supports custom commands (somewhat useful)

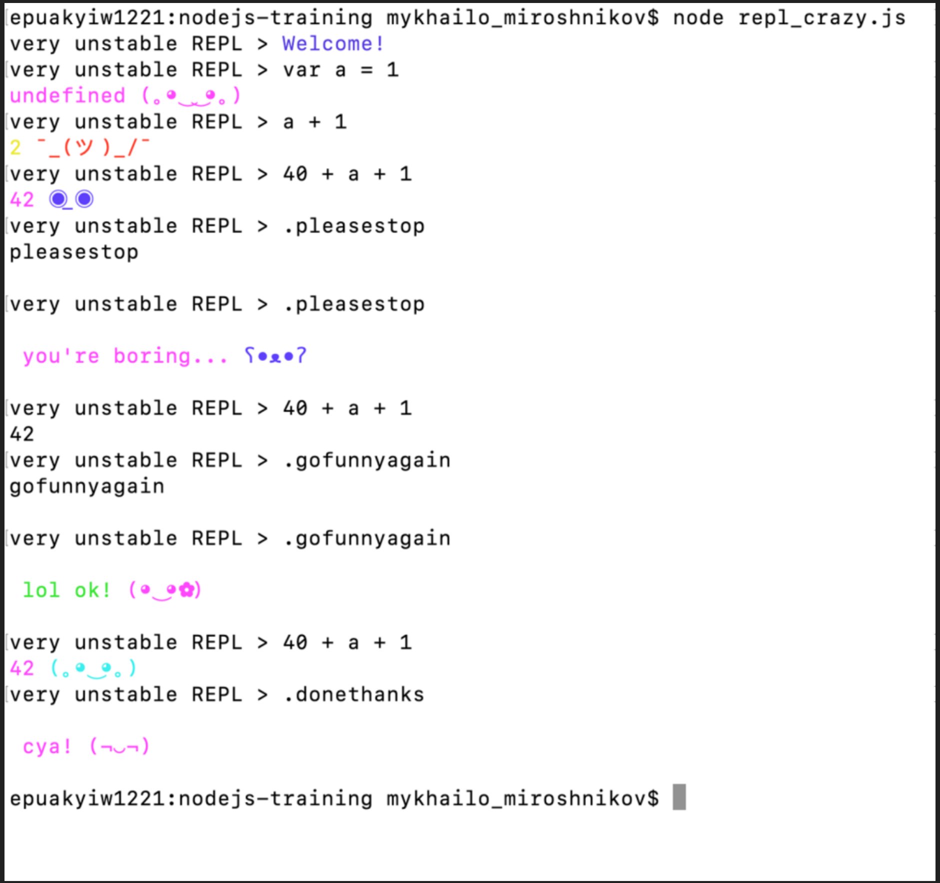
Global scope can be accessed/configured from both REPL environment and host process by using **REPLServer.context** (very useful)

Supports custom input/output streams - be it a server, DB, file or just a custom stream (very very useful)

Create your custom REPL

Create your own REPL with the help of core **`require(‘repl’)`** module

It supports various configuration options, up to the point of ultimate customization of experience.



```
epuakyiw1221:nodejs-training mykhailo_miroshnikov$ node repl_crazy.js
very unstable REPL > Welcome!
very unstable REPL > var a = 1
undefined (｡◕‿‿◕｡)
very unstable REPL > a + 1
2 ¯_(ツ)_/¯
very unstable REPL > 40 + a + 1
42 ◉_◉
very unstable REPL > .pleasestop
pleasestop

very unstable REPL > .pleasestop

 you're boring... ʕ•ᴥ•ʔ

very unstable REPL > 40 + a + 1
42
very unstable REPL > .gofunnyagain
gofunnyagain

very unstable REPL > .gofunnyagain

 lol ok! (◕‿◕✿)

very unstable REPL > 40 + a + 1
42 (｡◕‿◕｡)
very unstable REPL > .donethanks

 cya! (¬‿¬)

epuakyiw1221:nodejs-training mykhailo_miroshnikov$ █
```

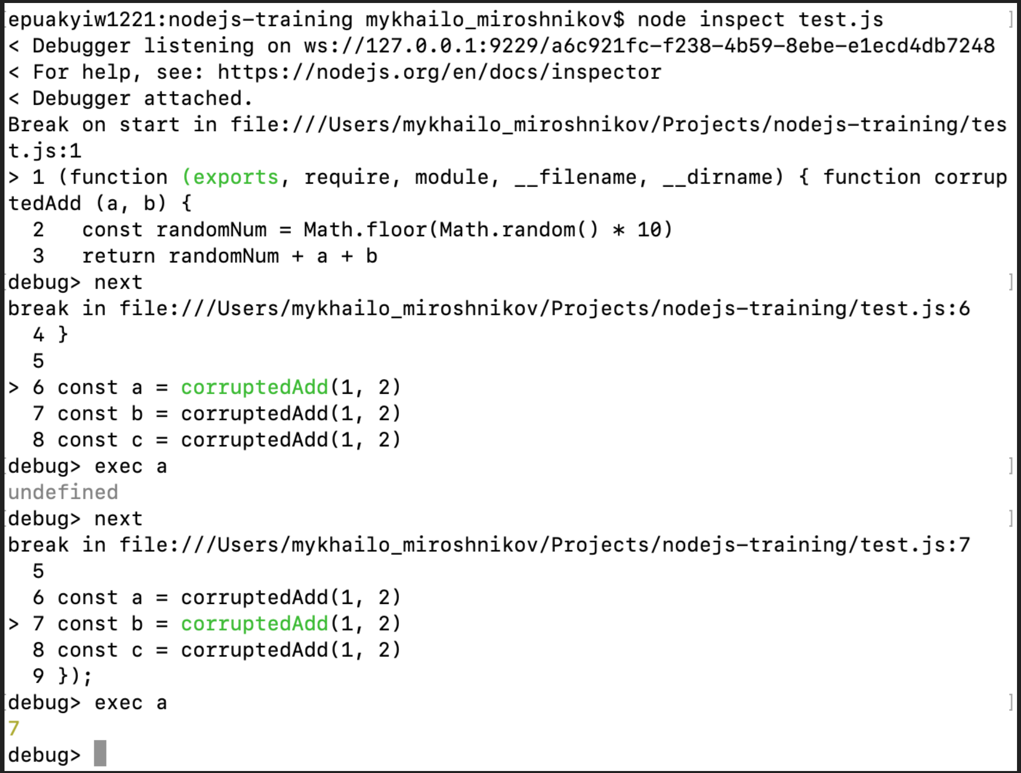
Debugging: REPL

`node inspect %file%`

Provides REPL for debugging

Has a number of useful commands

Almost never used, unless there’s no way to run in Chrome



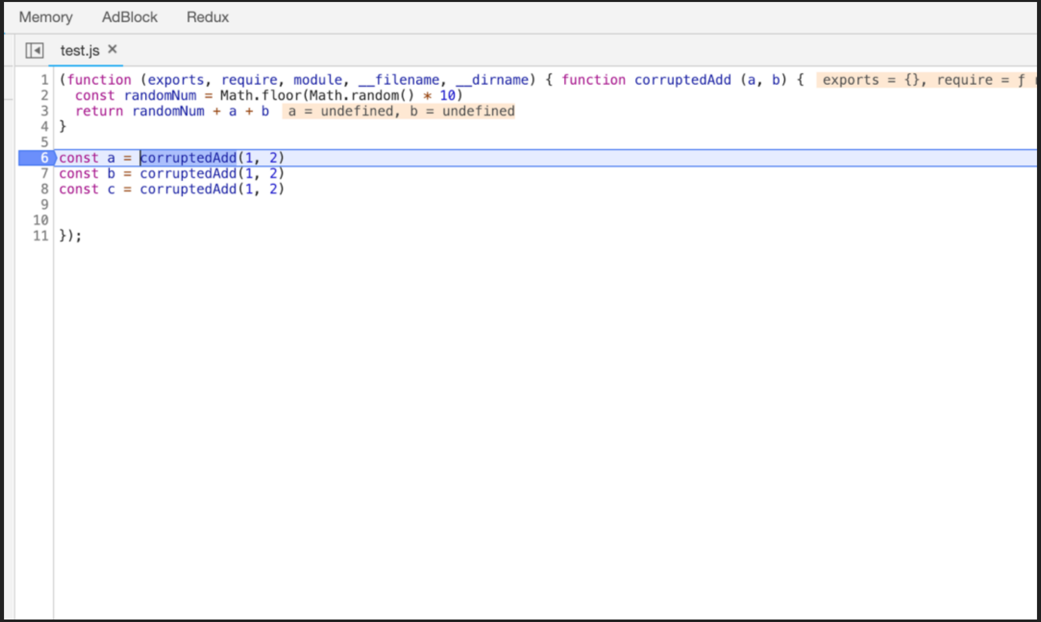
```
epuakyiw1221:nodejs-training mykhailo_miroshnikov$ node inspect test.js
< Debugger listening on ws://127.0.0.1:9229/a6c921fc-f238-4b59-8ebe-e1ecd4db7248
< For help, see: https://nodejs.org/en/docs/inspector
< Debugger attached.
Break on start in file:///Users/mykhailo_miroshnikov/Projects/nodejs-training/tes
t.js:1
> 1 (function (exports, require, module, __filename, __dirname) { function corrup
tedAdd (a, b) {
  2   const randomNum = Math.floor(Math.random() * 10)
  3   return randomNum + a + b
debug> next
break in file:///Users/mykhailo_miroshnikov/Projects/nodejs-training/test.js:6
  4 }
  5
> 6 const a = corruptedAdd(1, 2)
  7 const b = corruptedAdd(1, 2)
  8 const c = corruptedAdd(1, 2)
debug> exec a
undefined
debug> next
break in file:///Users/mykhailo_miroshnikov/Projects/nodejs-training/test.js:7
  5
  6 const a = corruptedAdd(1, 2)
> 7 const b = corruptedAdd(1, 2)
  8 const c = corruptedAdd(1, 2)
  9 });
debug> exec a
7
debug> 
```

Debugging: Chrome

`node --inspect-brk %file%`

`chrome://inspect`

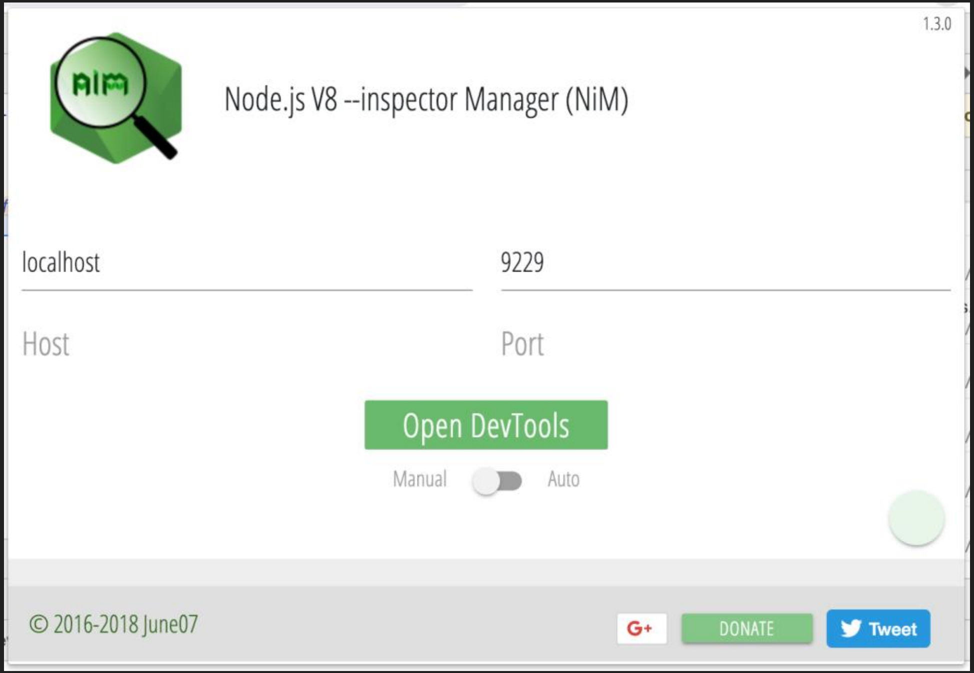
Allows to debug in Chrome



```
(function (exports, require, module, __filename, __dirname) { function corruptedAdd (a, b) {  exports = {}, require = ƒ
  const randomNum = Math.floor(Math.random() * 10)
  return randomNum + a + b  a = undefined, b = undefined
}

const a = corruptedAdd(1, 2)
const b = corruptedAdd(1, 2)
const c = corruptedAdd(1, 2)


});
```

Debugging: Chrome

To improve experience, you can install **Node Inspector Manager** extension for Chrome (will open debugger page as soon as inspector in started)


1.3.0

Node.js V8 --inspector Manager (NiM)

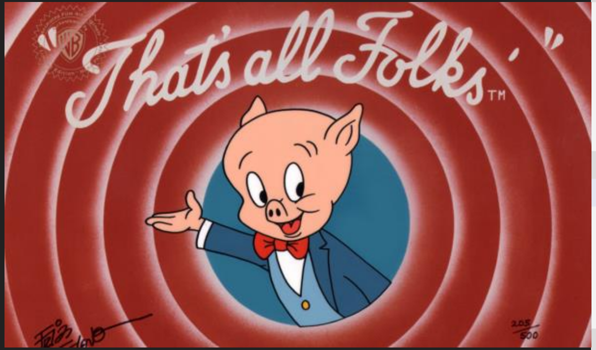
Host: localhost
Port: 9229

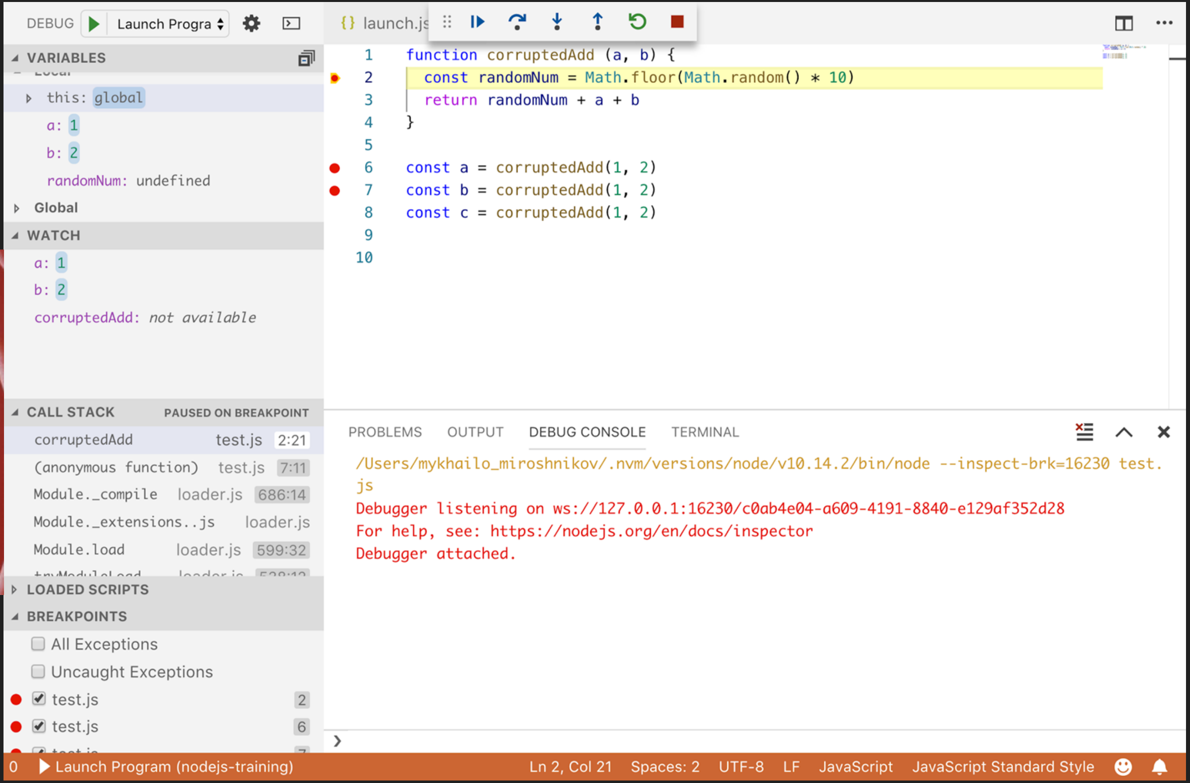
© 2016-2018 June07

Debugging: VSCode

Configure launch.json

Press F5





```
a: 1
b: 2
randomNum: undefined
```

```
a: 1
b: 2
correptedAdd: not available
```

```
corruptedAdd            test.js    2:21
(anonymous function)    test.js    7:11
Module._compile         loader.js  686:14
Module._extensions..js  loader.js
Module.load             loader.js  599:32
```

```
test.js  2
test.js  6
```

```javascript
function corruptedAdd (a, b) {
  const randomNum = Math.floor(Math.random() * 10)
  return randomNum + a + b
}

const a = corruptedAdd(1, 2)
const b = corruptedAdd(1, 2)
const c = corruptedAdd(1, 2)

```

```
/Users/mykhailo_miroshnikov/.nvm/versions/node/v10.14.2/bin/node --inspect-brk=16230 test.
js
Debugger listening on ws://127.0.0.1:16230/c0ab4e04-a609-4191-8840-e129af352d28
For help, see: https://nodejs.org/en/docs/inspector
Debugger attached.
```

Debugging: Remote

Set up SSH tunnel:

```
ssh –N –i -L 9229:127.0.0.1:9229
```

Set a message to target Node.js process to open debug port:

```
kill –SIGUSR1 %node_app_id%
```

Connect with Chrome Dev Tools

BE VERY CAREFUL **WITH PRODUCTION SERVERS**

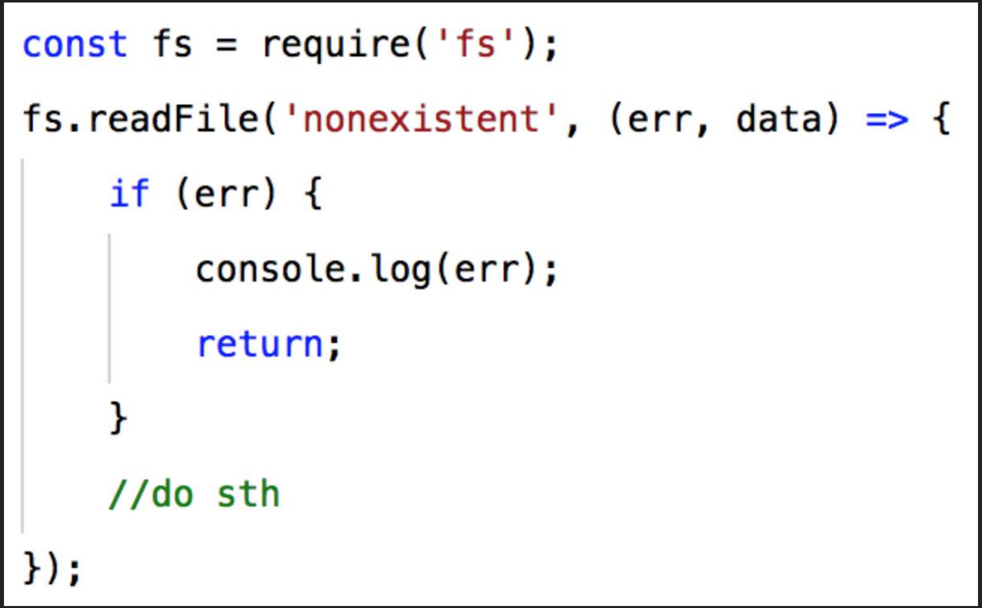
Handling Errors: How

- throw/try/catch/finally - rethrow when needed, **no empty catch()**
- Promise.catch
- async/await try/catch
- error-first callbacks
- EventEmitter errors

Optionally, create your own error types by extending from `Error`

Handling Errors: Error-first callback

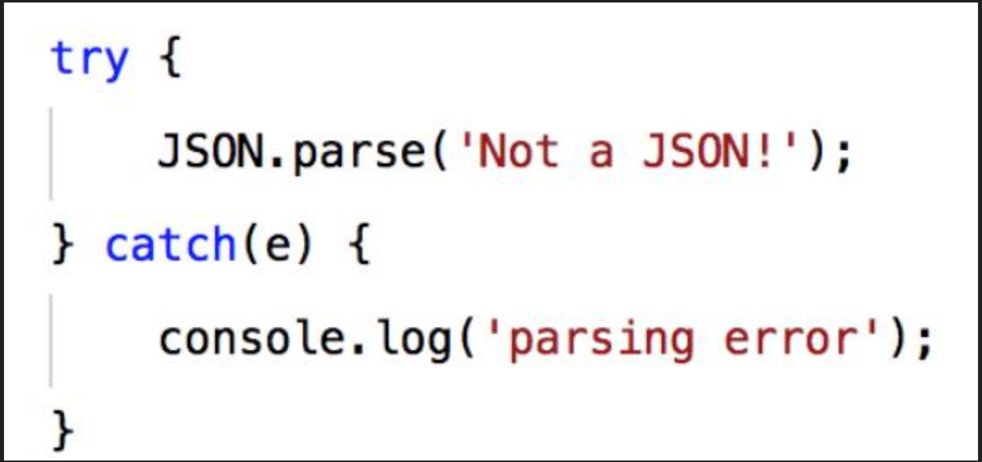
Common Node.JS approach to handle errors by propagating them up.



```
const fs = require('fs');

fs.readFile('nonexistent', (err, data) => {

    if (err) {

        console.log(err);

        return;

    }

    //do sth

});
```

Handling Errors: Try/catch

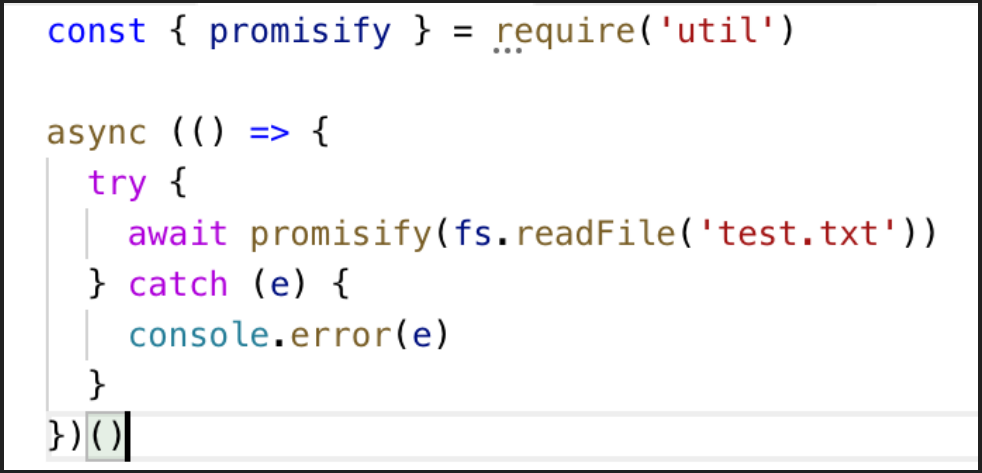
Common way to handle synchronous errors.



```
try {
    JSON.parse('Not a JSON!');
} catch(e) {
    console.log('parsing error');
}
```

Handling Errors: Async try/catch

Latest way to handle asynchronous errors.

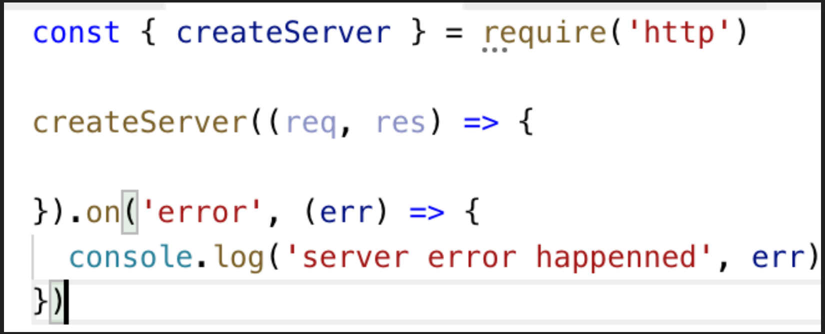


```
const { promisify } = require('util')

async (() => {
  try {
    await promisify(fs.readFile('test.txt'))
  } catch (e) {
    console.error(e)
  }
})()
```

Handling Errors: EventEmitter errors

Just another way to handle error in Node.JS

To throw an error use `EventEmitter.emit(‘error’, err)`



```
const { createServer } = require('http')

createServer((req, res) => {

}).on('error', (err) => {
  console.log('server error happenned', err)
})
```

Handling Errors: Common System Errors

- EACCES - permission denied
- EADDRINUSE - address already in use
- ECONNREFUSED - connection refused
- ENOENT – no such file or directory
- etc.

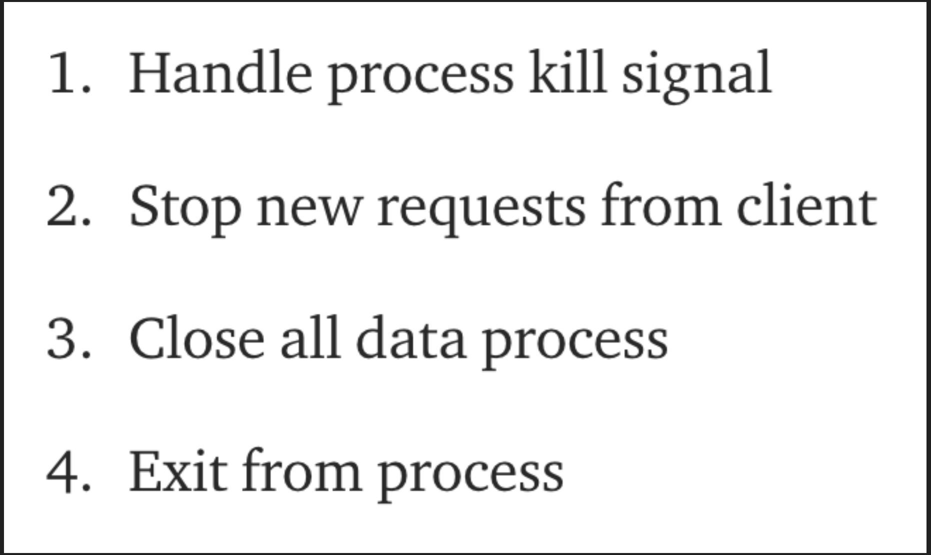
Handling Errors: Uncaught exceptions

```
process.on(‘uncaughtException’)

process.on(unhandledRejection’)
```

DO NOT RESTORE THE PROGRAM TO NORMAL OPERATION

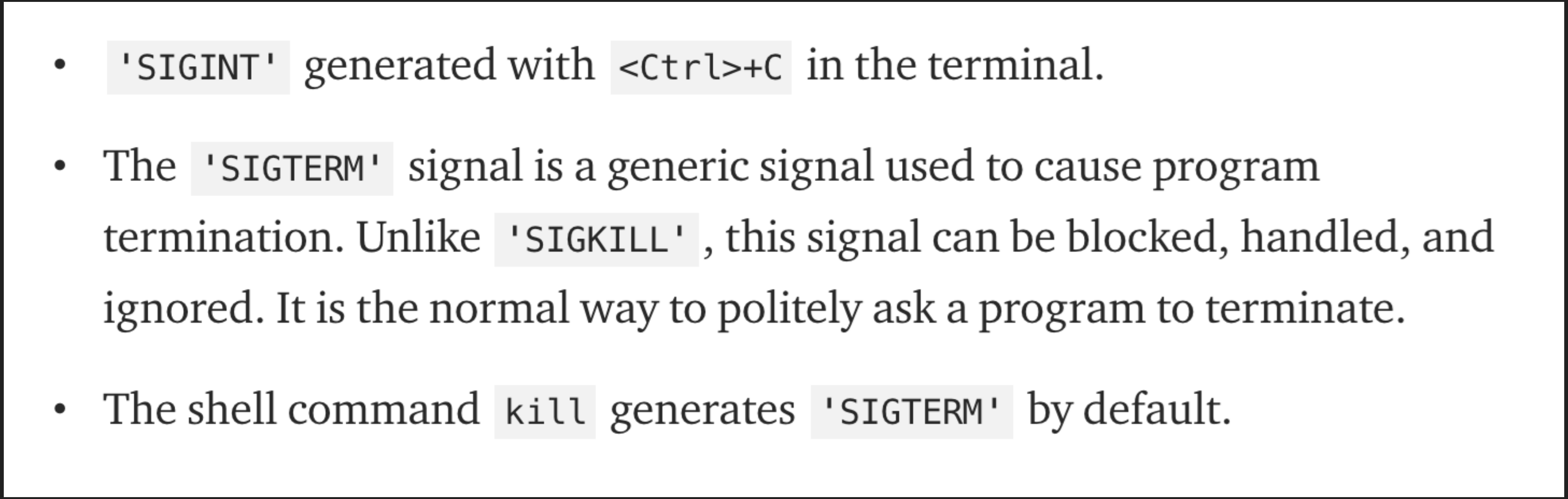
Log errors, free resources and `process.exit(1)`

Handling Errors: Shutdown gracefully



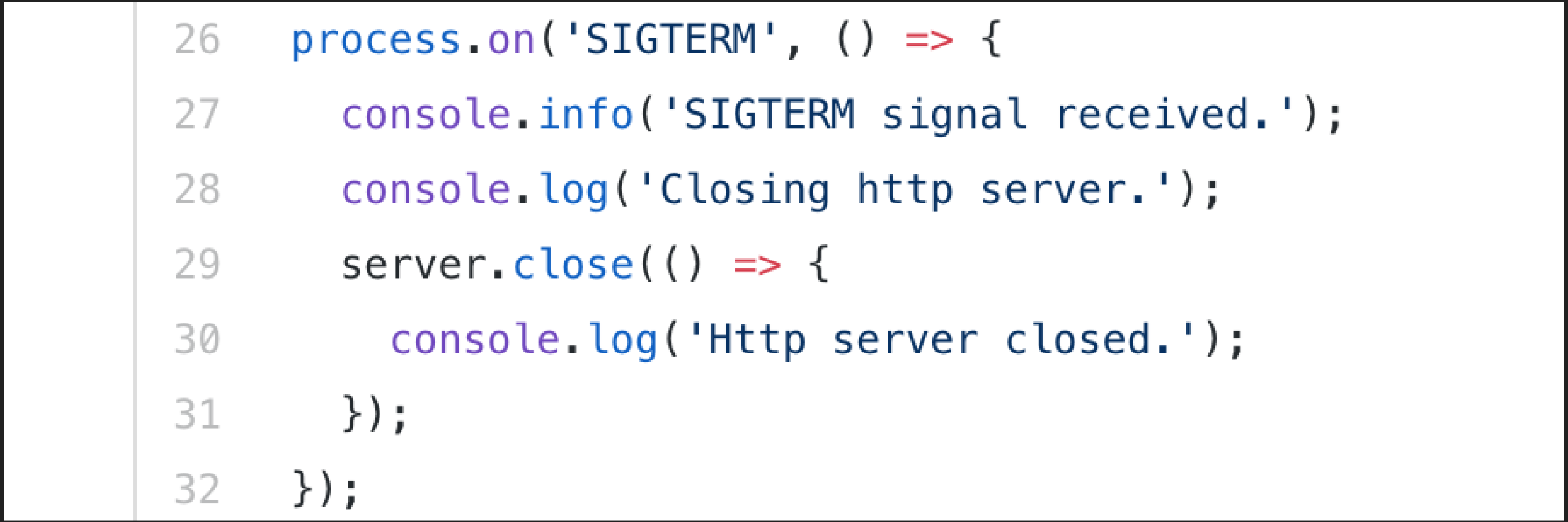
1. Handle process kill signal
2. Stop new requests from client
3. Close all data process
4. Exit from process

Graceful Shutdown: Listen for process kill signals



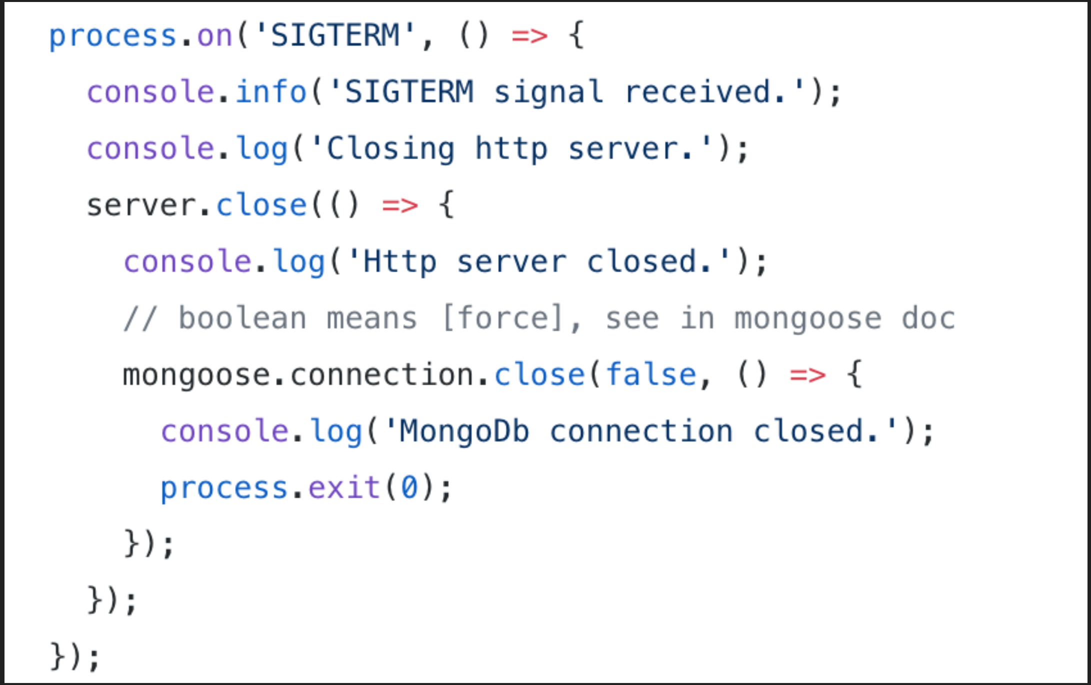
- `'SIGINT'` generated with `<Ctrl>+C` in the terminal.
- The `'SIGTERM'` signal is a generic signal used to cause program termination. Unlike `'SIGKILL'`, this signal can be blocked, handled, and ignored. It is the normal way to politely ask a program to terminate.
- The shell command `kill` generates `'SIGTERM'` by default.

Graceful Shutdown: Stop listening to new clients



```
process.on('SIGTERM', () => {
  console.info('SIGTERM signal received.');
  console.log('Closing http server.');
  server.close(() => {
    console.log('Http server closed.');
  });
});
```

Graceful Shutdown: Free resources and exit process



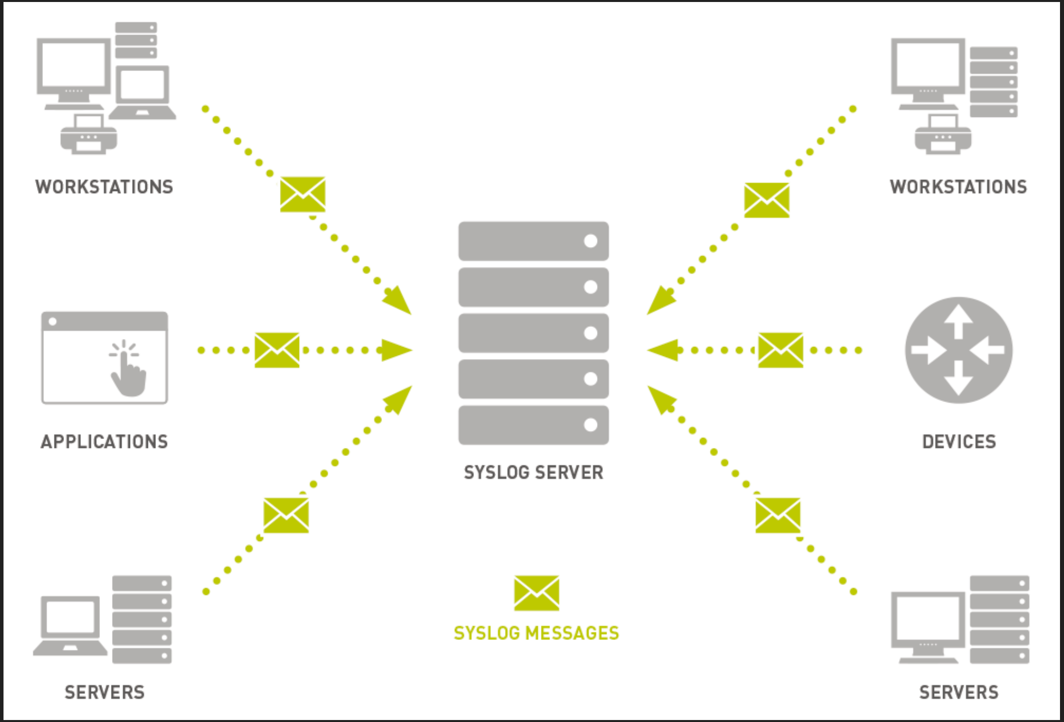
```
process.on('SIGTERM', () => {
  console.info('SIGTERM signal received.');
  console.log('Closing http server.');
  server.close(() => {
    console.log('Http server closed.');
    // boolean means [force], see in mongoose doc
    mongoose.connection.close(false, () => {
      console.log('MongoDb connection closed.');
      process.exit(0);
    });
  });
});
```

Logging: What to collect

- Errors
- Debug info
- Business stats (remember about **GDPR**)
- Other metrics (e.g. performance)

Logging: Syslog Protocol

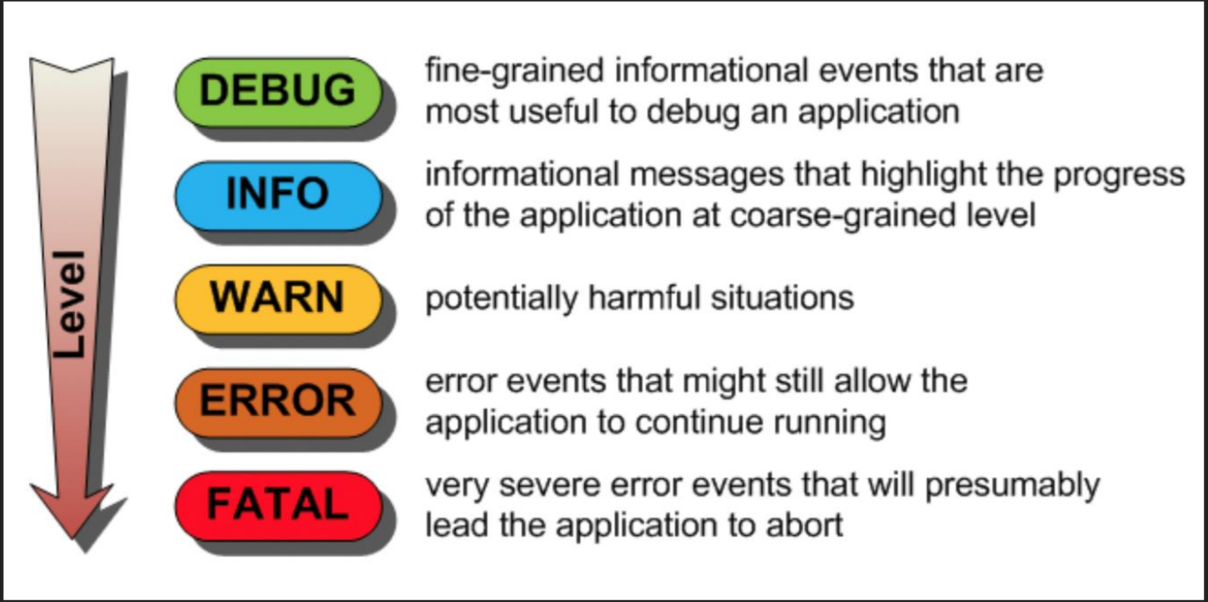
The protocol by which logs are generated, transported and stored.



Logging: Levels

NPM Levels:

0: error
1: warn
2: info
3: verbose
4: debug
5: silly



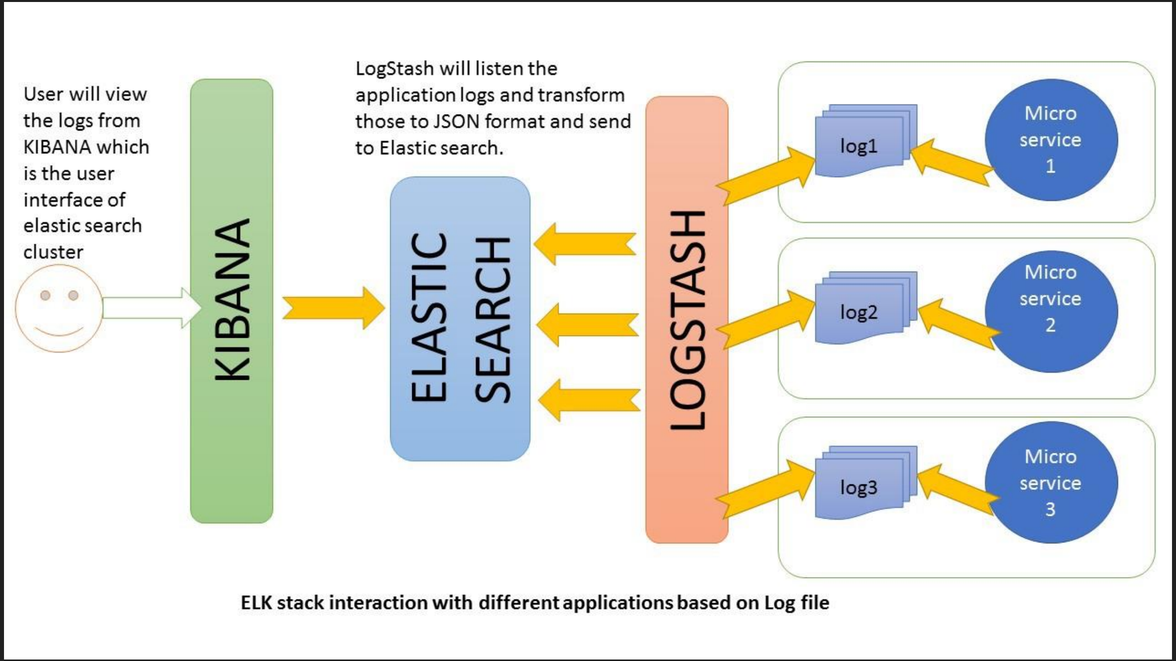
DEBUG	fine-grained informational events that are most useful to debug an application
INFO	informational messages that highlight the progress of the application at coarse-grained level
WARN	potentially harmful situations
ERROR	error events that might still allow the application to continue running
FATAL	very severe error events that will presumably lead the application to abort

Logging: ELK Stack

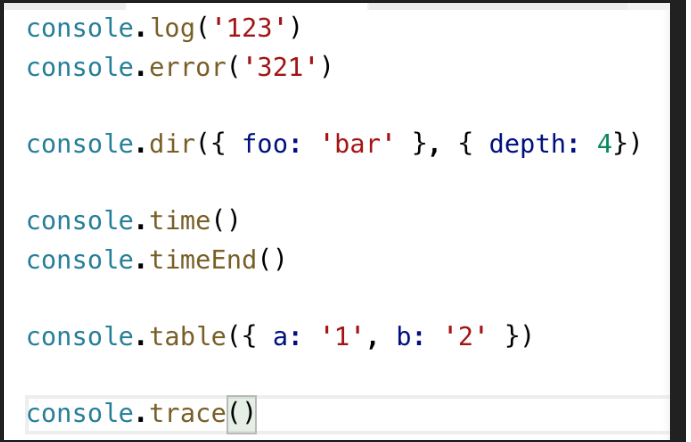
Elasticsearch is a NoSQL database

Logstash is a log pipeline tool that accepts inputs from various sources

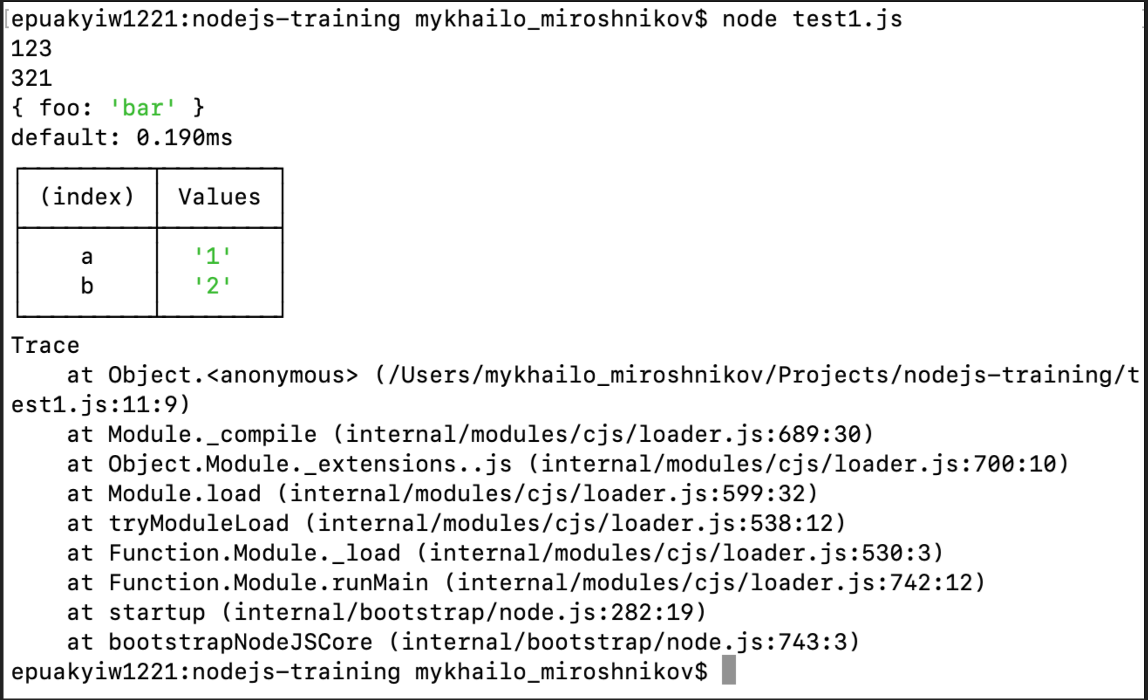
Kibana is a visualization layer that works on top of Elasticsearch


User will view the logs from KIBANA which is the user interface of elastic search cluster
KIBANA
LogStash will listen the application logs and transform those to JSON format and send to Elastic search.
ELASTIC SEARCH
LOGSTASH
log1
log2
log3
Micro service 1
Micro service 2
Micro service 3
ELK stack interaction with different applications based on Log file

Logging: Console



```
console.log('123')
console.error('321')

console.dir({ foo: 'bar' }, { depth: 4})

console.time()
console.timeEnd()

console.table({ a: '1', b: '2' })

console.trace()
```

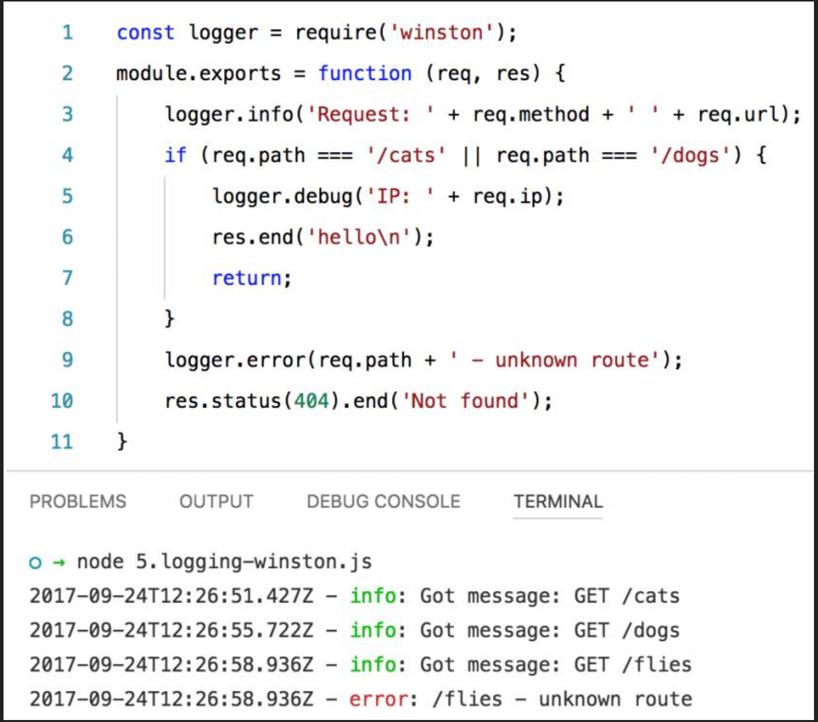


```
epuakyiw1221:nodejs-training mykhailo_miroshnikov$ node test1.js
123
321
{ foo: 'bar' }
default: 0.190ms
┌─────────┬────────┐
│ (index) │ Values │
├─────────┼────────┤
│    a    │  '1'   │
│    b    │  '2'   │
└─────────┴────────┘
Trace
    at Object.<anonymous> (/Users/mykhailo_miroshnikov/Projects/nodejs-training/t
est1.js:11:9)
    at Module._compile (internal/modules/cjs/loader.js:689:30)
    at Object.Module._extensions..js (internal/modules/cjs/loader.js:700:10)
    at Module.load (internal/modules/cjs/loader.js:599:32)
    at tryModuleLoad (internal/modules/cjs/loader.js:538:12)
    at Function.Module._load (internal/modules/cjs/loader.js:530:3)
    at Function.Module.runMain (internal/modules/cjs/loader.js:742:12)
    at startup (internal/bootstrap/node.js:282:19)
    at bootstrapNodeJSCore (internal/bootstrap/node.js:743:3)
epuakyiw1221:nodejs-training mykhailo_miroshnikov$
```

Logging: Winston

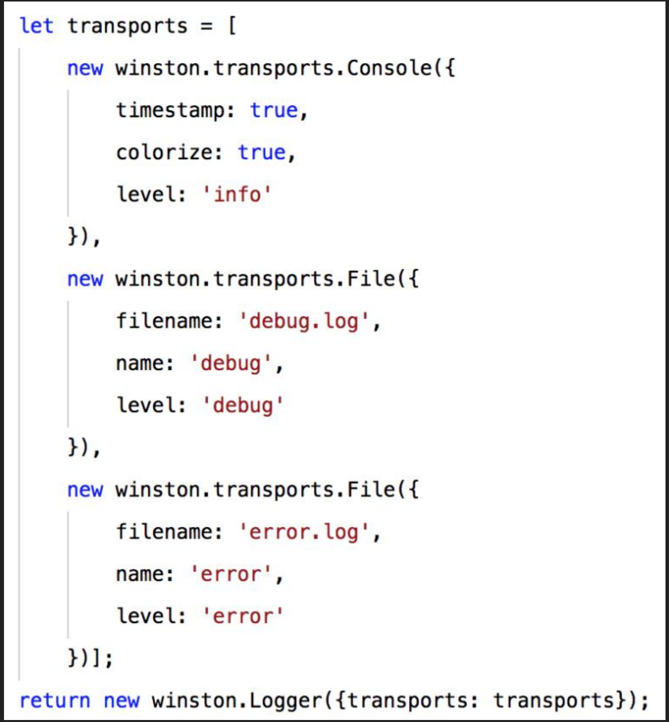
- Logging to, Console, Files, Databases
- Logging levels
- Timestamps
- Configurable output
- Log rotation
- String/JSON

Logging: Winston



```
const logger = require('winston');
module.exports = function (req, res) {
    logger.info('Request: ' + req.method + ' ' + req.url);
    if (req.path === '/cats' || req.path === '/dogs') {
        logger.debug('IP: ' + req.ip);
        res.end('hello\n');
        return;
    }
    logger.error(req.path + ' - unknown route');
    res.status(404).end('Not found');
}
```

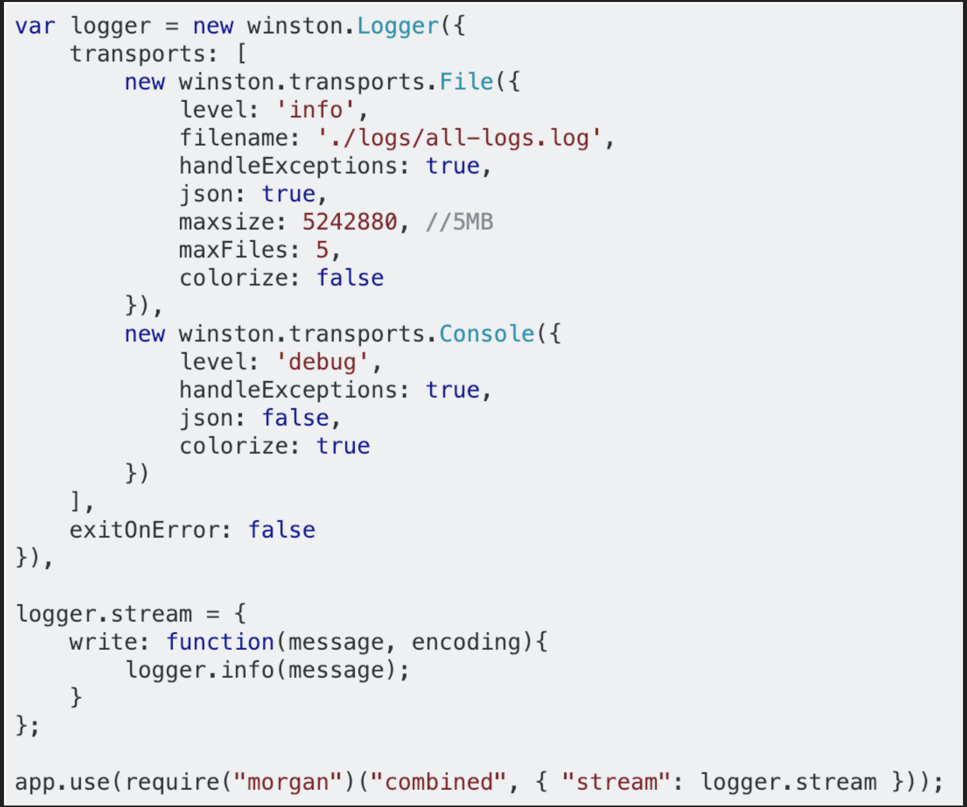
```
o → node 5.logging-winston.js
2017-09-24T12:26:51.427Z - info: Got message: GET /cats
2017-09-24T12:26:55.722Z - info: Got message: GET /dogs
2017-09-24T12:26:58.936Z - info: Got message: GET /flies
2017-09-24T12:26:58.936Z - error: /flies - unknown route
```



```
let transports = [
    new winston.transports.Console({
        timestamp: true,
        colorize: true,
        level: 'info'
    }),
    new winston.transports.File({
        filename: 'debug.log',
        name: 'debug',
        level: 'debug'
    }),
    new winston.transports.File({
        filename: 'error.log',
        name: 'error',
        level: 'error'
    })];
return new winston.Logger({transports: transports});
```

Logging: Ultimate

Combine Winston and Morgan to achieve the ultimate experience



```
var logger = new winston.Logger({
    transports: [
        new winston.transports.File({
            level: 'info',
            filename: './logs/all-logs.log',
            handleExceptions: true,
            json: true,
            maxsize: 5242880, //5MB
            maxFiles: 5,
            colorize: false
        }),
        new winston.transports.Console({
            level: 'debug',
            handleExceptions: true,
            json: false,
            colorize: true
        })
    ],
    exitOnError: false
}),

logger.stream = {
    write: function(message, encoding){
        logger.info(message);
    }
};

app.use(require("morgan")("combined", { "stream": logger.stream }));
```