
```
fs.readFile    fs.readFileSync
```

```
.toString('utf8')
```

```
process.stdin    fs.createReadStream  http.IncomingMessage
```

```
fs.createWriteStream http.ServerResponse process.stdout
```

```
zlib.createGzip()
```

```
data
```

```

false
write()
end()
write()

```



```
callback()  
callback(err)
```

```
pipe()  
stream/promises
```

```
pipeline()
```





`pipe()` `pipeline()`



Buffer	Buffer.alloc(n) Buffer.from() .toString()	
	fs.createReadStream() on('data') for await	
	fs.createWriteStream() write() end()	
	extends Transform _transform(chunk, enc, cb)	
pipe()	readable.pipe(writable)	
pipeline()	await pipeline(r, t, w)	
	write() false 'drain'	pipe() pipeline()

```
    pipeline()    'error'  
  
    callback()    callback(err)  
  
                                'data'  
  
    for await...of
```

```
countLines(filePath)    fs.readFile
```

CensorTransform

Transform

process.stdout

compressFile(inputPath)

pipeline()

inputPath + '.gz'

```
readable.on('data', ...)  
writable.on('error', ...) server.on('request', ...)
```

```
emit('eventName', ...args)
```

```
on('eventName', fn)
```

```
once('eventName', fn)  
'connect' 'open' 'ready'
```

```
off('eventName', fn)      removeListener  
removeAllListeners('eventName')
```



`emit()`

`setImmediate()` `process.nextTick()`





`'error'`

`'error'`



`on()`



```
prependListener('event', fn)
```

```
fs.readFile(path, cb)
```

```
await fs.promises.readFile(path)
```

```
stream.on('data', ...)
```

```
server.on('request', ...)
```

```
events.once()
```

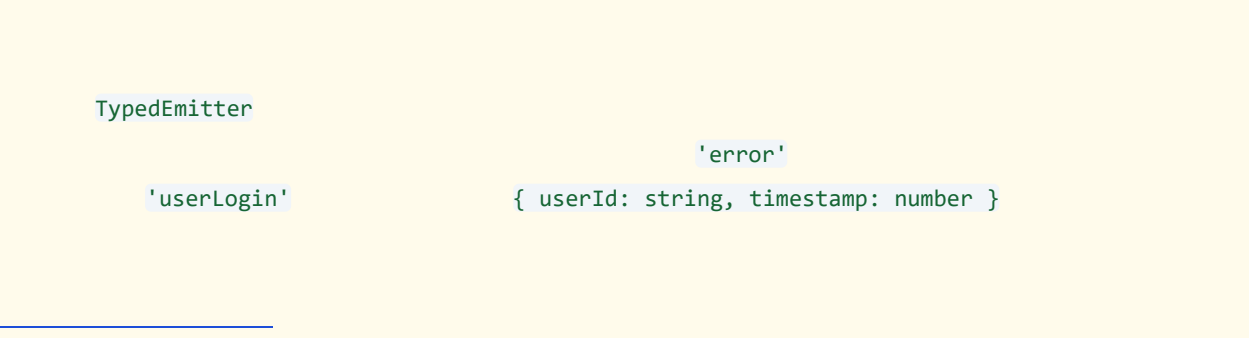
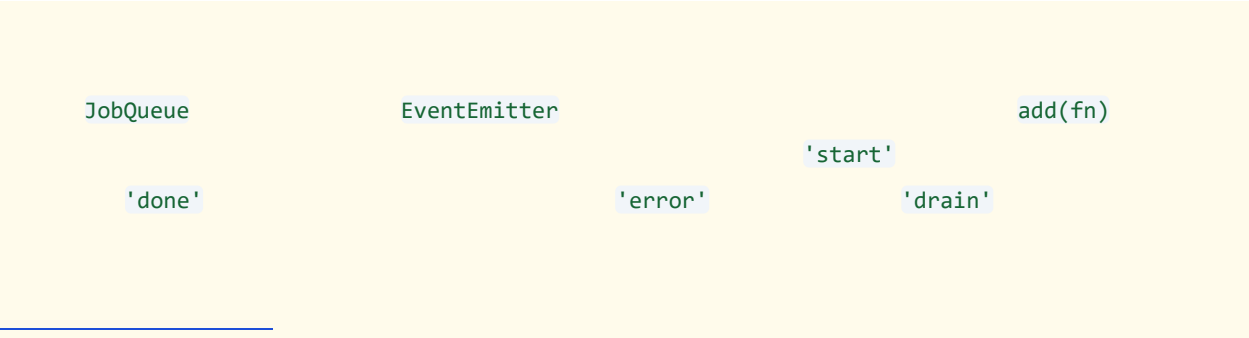
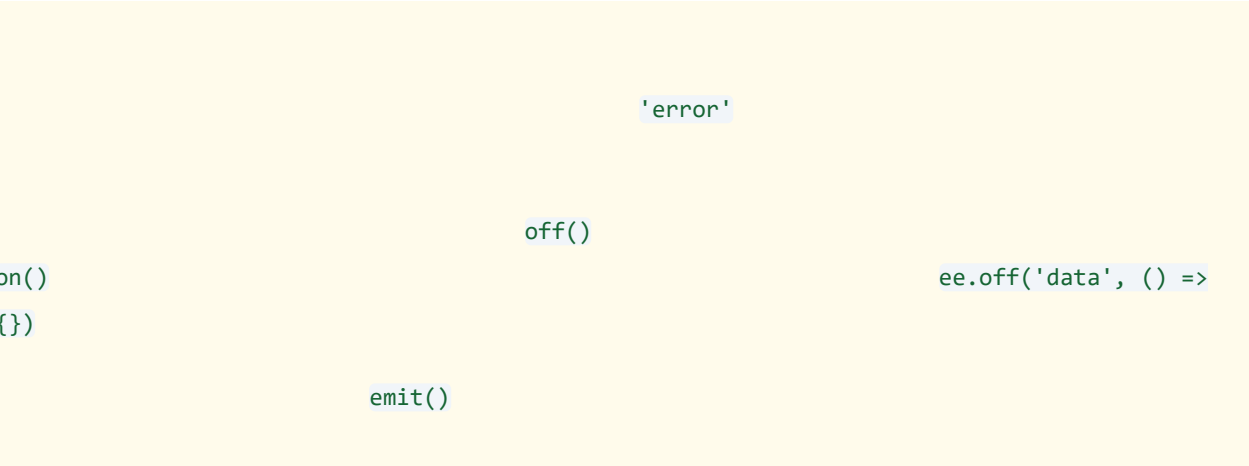
```
on(event, fn)
```

```
once(event, fn)
```

```
emit(event, ...args)
```

```
off(event, fn)
```

removeAllListeners(event)	
listenerCount(event)	
eventNames()	
setMaxListeners(n)	
prependListener(event, fn)	
events.once(emitter, event)	





fork

exec execFile spawn

/bin/sh cmd.exe

exec

stdout stderr

spawn

.send() 'message'

exec()
stdout, stderr)

(error,

```
exec()  
ERR_CHILD_PROCESS_STDIO_MAXBUFFER      maxBuffer      spawn()  
  
exec('command', { maxBuffer: 10 * 1024 * 1024 }, callback)
```

```
execFile()
```



`spawn()` `ChildProcess` `stdout` `stderr`





`fork()`

`.send()`

`on('message')`





```
fork()  
  
    fork()
```



exec()

```
exec(`grep ${userInput} /var/log/app.log`, cb)
```

```
userInput    foo; rm -rf /
```

```
execFile()  spawn()
```

exec()

execFile()

spawn()

fork()

```
'close'
```

```
'error'
```

```
send()
```

```
runWithTimeout(command, args, timeoutMs)
```

```
{ stdout, code }
```

```
timeoutMs
```

```
TimeoutError
```

```
AbortController
```

```
TaskFarm
```

```
N
```

```
worker.js
```

```
run(data)
```

```
shutdown()
```

```
taillog(filePath, lines)
```

```
spawn()
```

```
tail -f
```

```
'line'
```

```
stop()
```

SharedArrayBuffer

true

false

MessagePort

.on('message')

null

.postMessage()

{ workerData: ... }

.postMessage()

on('message') on('error')

on('exit')

isMainThread



```
__filename  
  
Worker  
  
new Worker('./worker.js', { workerData: { ... } })
```



postMessage

SharedArrayBuffer



Atomics

Atomics

TypedArray

SharedArrayBuffer



ArrayBuffer

SharedArrayBuffer



MessageChannel

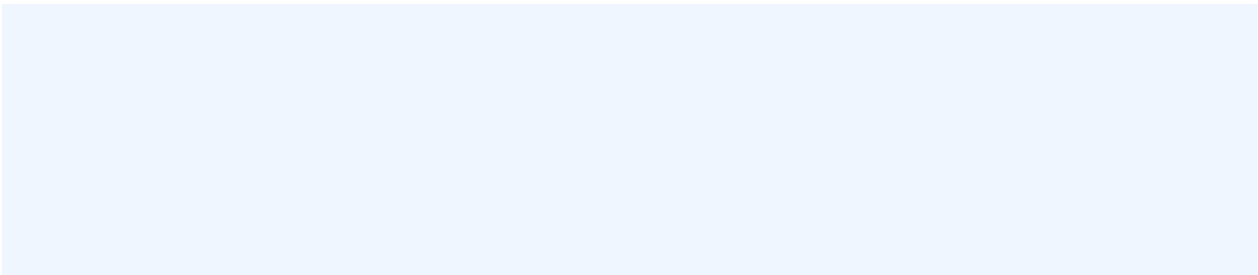


TaskFarm





	SharedArrayBuffer		
--	-------------------	--	--



```
postMessage      SharedArrayBuffer

Atomics.wait()      Atomics.waitAsync()
```

```
parallelMap(array, workerFile, concurrency)      concurrency

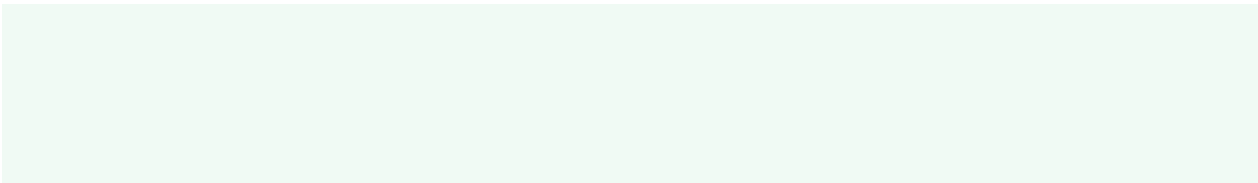
{ chunk }      workerData
```

```
sharedArray[0]++

Atomics.add(sharedArray, 0, 1)
```

```
WorkerPool

pool.run(payload, onProgress)
  { type: 'progress', taskId, percent }      { type:
'done', taskId, result }
```



`mysql2/promise`

`execute()`

`mysql`

`mysql` `mysql2`





```
dotenv                                process.env.DB_PASSWORD                .env
```

```
                                execute()
```

```
query(sql,  
values)
```

```
execute(sql,  
values)
```



`query()`

`execute()`

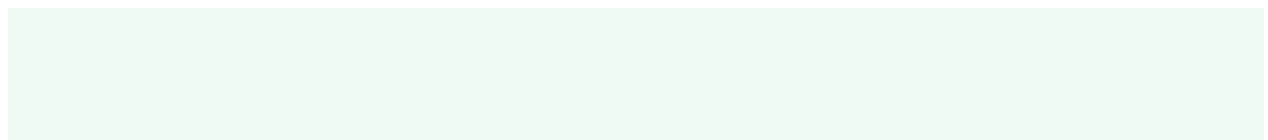
`[rows, fields]`



?



```
const ALLOWED = ['name', 'email', 'created_at'];  
if (!ALLOWED.includes(col)) throw new Error('Invalid column');  
conn.execute(`SELECT ${col} FROM users WHERE id = ?`, [id])
```





db.js

```
// db.js
const pool = mysql.createPool({ ... });
module.exports = pool;

// anywhere else
const pool = require('./db');
const [rows] = await pool.execute('SELECT ...');
```





```
pool.execute()
```

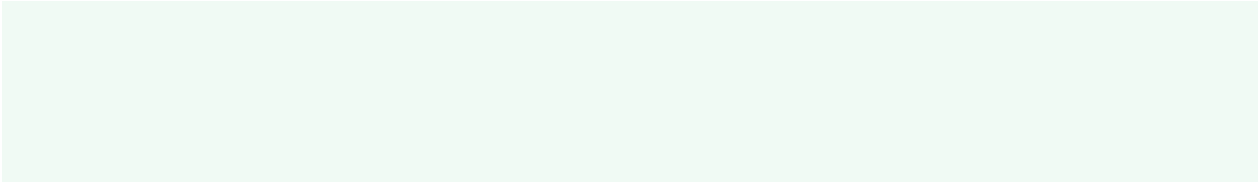


```
query()                                execute()                                execute()
                                     release()
```

```
UserRepository                                findById(id) findById(email)
create({ name, email, passwordHash }) update(id, fields) fields
delete(id) update()                                null
```

```
placeOrder(userId, items) items { productId, quantity }
order_items
```

```
searchUsers(options) { query, page, pageSize, orderBy, orderDir }
{ data, total, page, pageSize, totalPages } name
email LIKE orderBy Promise.all
```



process.env



dotenv .env

process.env





PORT

dotenv

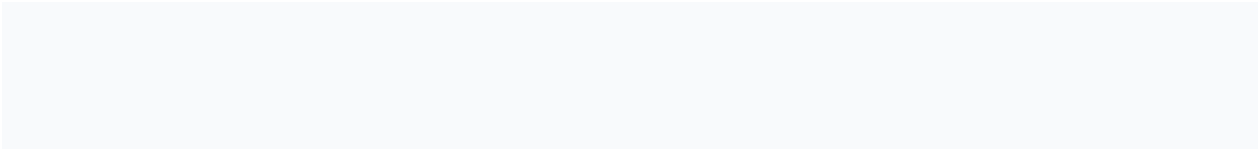
.env

```
require('dotenv').config({ override: true })
```

.gitignore .env

.env

```
dotenv.config({ path: '.env.test' })
```



`process.env.SOMETHING`
`config.js`



JWT_SECRET

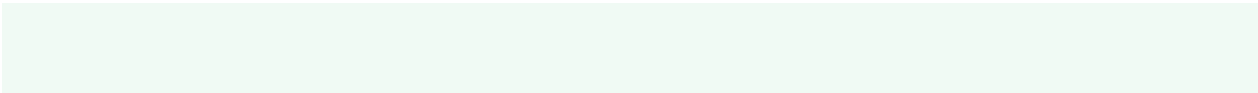


```
npm install envalid

const { cleanEnv, str, port, num } = require('envalid');
const env = cleanEnv(process.env, {
  JWT_SECRET: str({ docs: 'Generate with crypto.randomBytes(64)' }),
  PORT: port({ default: 3000 }),
  DB_NAME: str(),
});
```

NODE_ENV

development		
test		
production		



```
process.env
```

```
console.log(config)    process.env
```

```
"start": "JWT_SECRET=abc123 node server.js"
```



```

                                dotenv.config()

                                process.env.config()

                                process.env.FLAG = 'false'                                'false'
                                === 'true'

                                process.env.NODE_ENV == 'production'
                                ' production'                                === 'production'

isProd

```

```

                                validateConfig(schema, env)
process.env                                required                                type 'string' 'number' 'boolean'
'port' default                                min max

```

```

                                loadConfig()
.env.{NODE_ENV}                                .env.test                                .env

                                fs.readFileSync                                KEY=VALUE

```

```

                                createSafeLogger(sensitiveKeys)                                info warn                                error

sensitiveKeys                                '[REDACTED]'

                                process.env

```



```
src/users.js  src/users.test.js  
src/users.js  __tests__/users.test.js
```





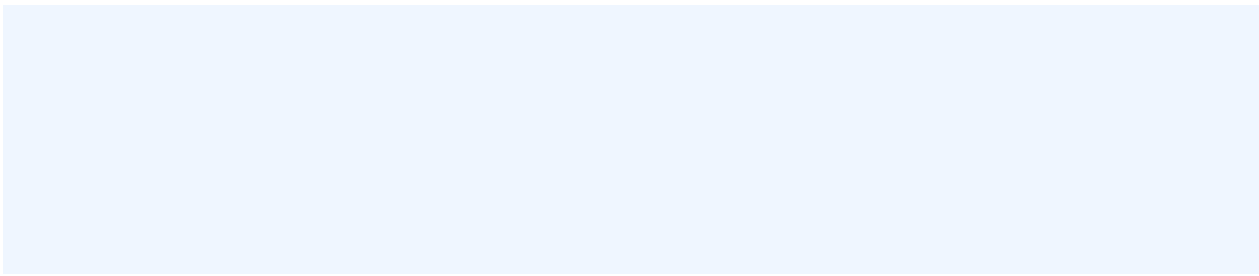














beforeEach afterEach

jest.clearAllMocks() afterEach
toHaveBeenCalledTimes

expect(promise).rejects.toThrow()

await

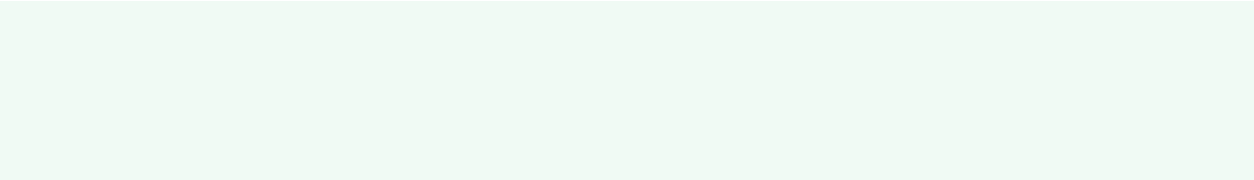
passwordUtils.js hashPassword(plain)
verifyPassword(plain, hash)

npm install bcrypt

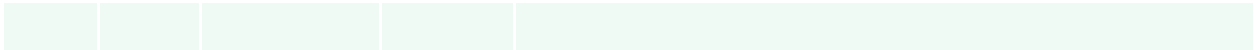
weatherService.js

fetch fetch jest.spyOn(global, 'fetch')
NetworkError NotFoundError

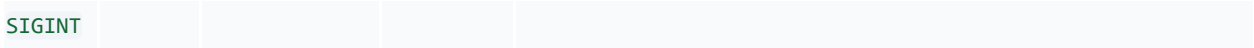
```
RateLimiter      N      windowMs
check(key)      { allowed: boolean, remaining: number, resetAt: number }
                (N+1)
                windowMs      jest.useFakeTimers()      jest.advanceTimersByTime()
```



Ctrl+C

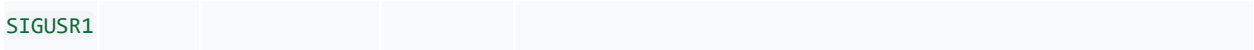


SIGTERM



SIGINT

SIGHUP



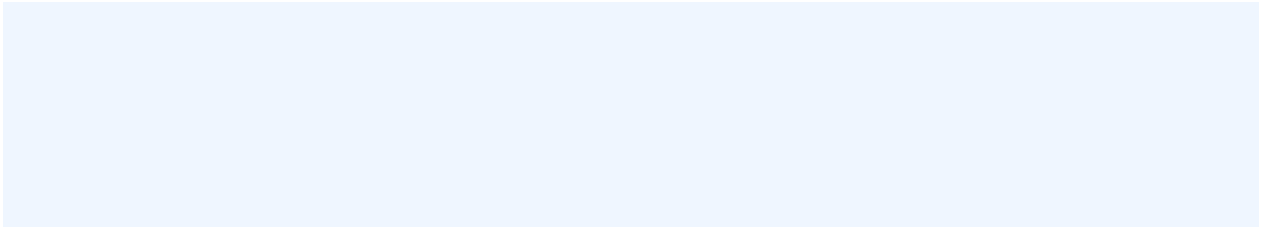
SIGUSR1

SIGKILL



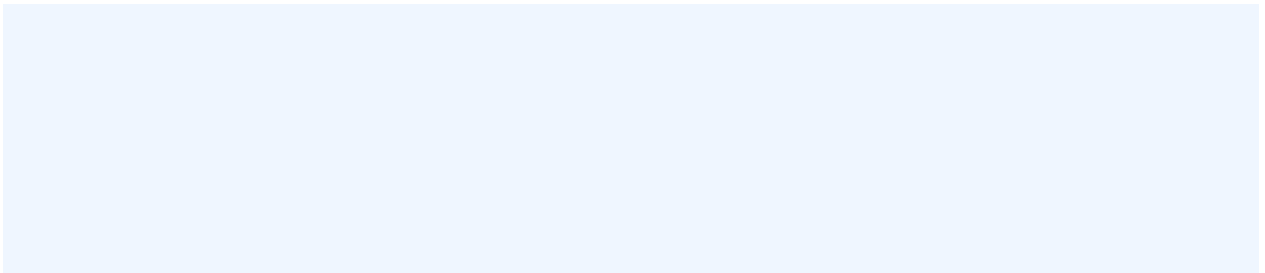


`try/catch` `.catch()`



`cluster`



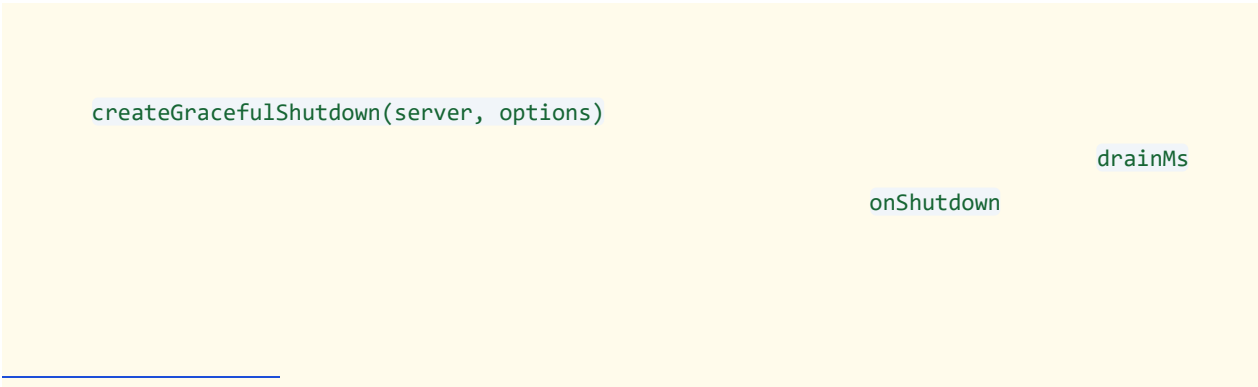
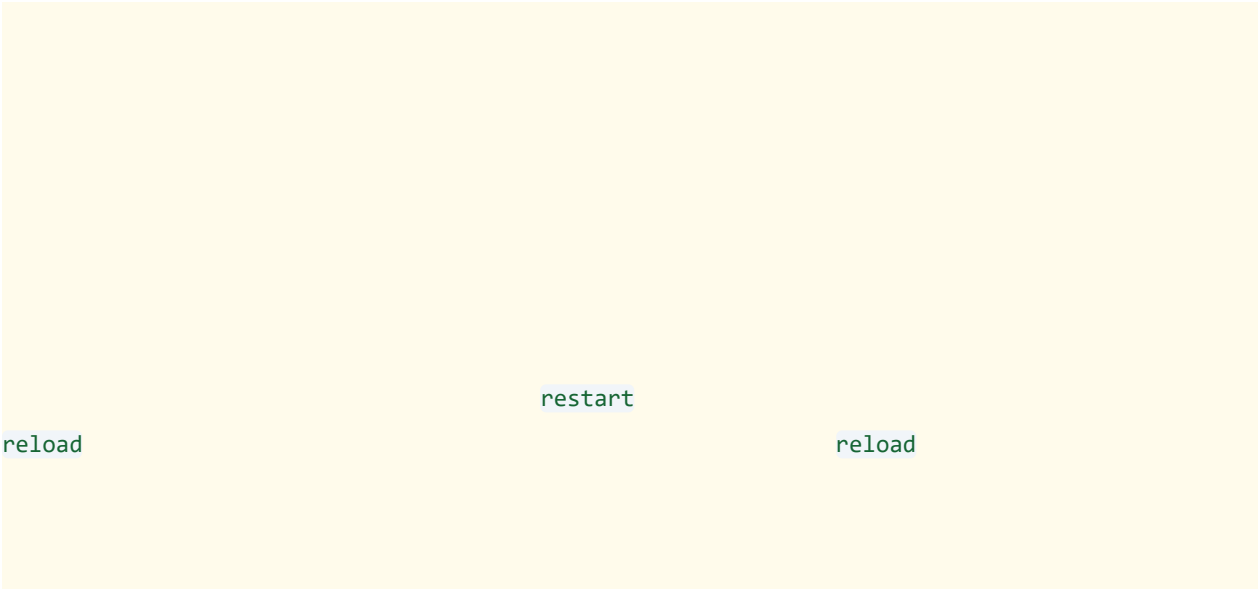






`pm2 reload`

`SIGINT`



```
GET /health
```

```
kill -HUP <pid>
```
