Node.js Process

Each Node.js script that runs is, in essence, a process. For example, `ps aux | grep 'node'` outputs all Node.js programs running on a machine. Conveniently, developers can access useful process information in code with the `process` object, e.g., `node -e "console.log(process.pid)"` will print the process ID.

Other useful `process` information includes:

- `env`: Environment variables
- `argv`: Command-line arguments
- `exit()`: Method to exit/terminate process

Let's see how to use each of them.

Environment Variables

Environment variables can be accessed via the `env` attribute:

```
console.log(process.env)

{ SHELL: '/bin/bash',
  USER: 'jordan',
  HOME: '/home/jordan',
  ...
}
```

A short one-liner can set the environment variable in bash, and then run Node eval to print the value. This is a bash/Terminal/ Command Prompt command which will print “development”:

```
NODE_ENV=development node -e "console.log(process.env.NODE_ENV)"
```

NODE_ENV is a convention. Common values include:

- development: used by developers to code verbose error messages and logs for debugging
- production: used by developers to hides excessive error messages and logs

This is just a convention but some libraries and frameworks will augment their behavior to hide error messages, e.g., Express.

Command-Line Arguments

To access CLI arguments, use the `process.argv` property which is an array.

For example, if the command is

```
node app.js arg1 arg2 arg3=val3
```

The first two elements are ‘node’ and the application's name while the rest are the command-line arguments. Thus, `process.argv`:

```
[
  'node',
  'app.js',
  'arg1',
  'arg2',
  'arg3=val3'
]
```

Exiting a Process

To exit a process, use the `exit` function:

```
process.exit()
```

When your application encounters an error, you want to exit with errors. Exit codes can also be specified

```
// this process failed
process.exit(1)

// this process failed with a different code
process.exit(129)

// this process exits successfully
process.exit(0)
```

Different failure codes can be used to differentiate types of failure. And knowing how an application failed allows the developers the means to program an appropriate response.