nodemon

reloads automatically

```
npm install -g nodemon
```

Nodemon is a utility **depended on about 3 million projects**, that will monitor for any changes in your source and automatically restart your server. Perfect for development.

Swap **nodemon** instead of **node** to run your code, and now your process will automatically restart when your code changes. To install, get Node.js, then from your terminal run:

```
npm install -g nodemon
```

Features

- Automatic restarting of application.
- Detects default file extension to monitor.
- Default support for node but easy to run any executable, such as python, ruby, make, etc.
- Ignoring specific files or directories.
- Watch specific directories.
- Works with server applications or one time run utilities and REPLs.
- Scriptable through node require statements.
- Open source and available on github.

Read the full documentation or visit the FAQ

Documentation

About nodemon

nodemon is a tool that helps develop Node.js based applications by automatically restarting the node application when file changes in the directory are detected.

nodemon does not require any additional changes to your code or method of development. nodemon is a replacement wrapper for node. To use nodemon, replace the word node on the command line when executing your script.

Installation

You can install either through cloning with git or by using npm (recommended).

using npm:

```
npm install -g nodemon
```

using yarn:

```
yarn global add nodemon
```

And nodemon will be installed globally to your system path.

You can also install nodemon as a development dependency:

using npm:

```
npm install --save-dev nodemon
```

using yarn:

```
yarn add nodemon -D
```

With a local installation, nodemon will not be available in your system path or you can't use it directly from the command line. Instead, the local installation of nodemon can be run by calling it from within an npm script (such as npm start) or using npx nodemon.

Usage

nodemon wraps your application, so you can pass all the arguments you would normally pass to your app:

```
nodemon [your node app]
```

For CLI options, use the -h (or --help) argument:

```
nodemon -h
```

Using nodemon is simple, if my application accepted a host and port as the arguments, I would start it as so:

```
nodemon ./server.js localhost 8080
```

Any output from this script is prefixed with [nodemon], otherwise all output from your application, errors included, will be echoed out as expected.

You can also pass the inspect flag to node through the command line as you would normally:

```
nodemon --inspect ./server.js 80
```

If you have a package.json file for your app, you can omit the main script entirely and nodemon will read the package.json for the main property and use that value as the app (ref).

nodemon will also search for the scripts.start property in package.json (as of nodemon 1.1.x).

Also check out the FAQ or issues for nodemon.

Automatic re-running

nodemon was originally written to restart hanging processes such as web servers, but now supports apps that cleanly exit. If your script exits cleanly, nodemon will continue to monitor the directory (or directories) and restart the script if there are any changes.

Manual restarting

Whilst nodemon is running, if you need to manually restart your application, instead of stopping and restart nodemon, you can type rs with a carriage return, and nodemon will restart your process.

Config files

nodemon supports local and global configuration files. These are usually named nodemon.json and can be located in the current working directory or in your home directory. An alternative local configuration file can be specified with the --config <file> option.

The specificity is as follows, so that a command line argument will always override the config file settings:

- command line arguments
- local config
- global config

A config file can take any of the command line arguments as JSON key values, for example:

```
{
  "verbose": true,
  "ignore": ["*.test.js", "**/fixtures/**"],
  "execMap": {
    "rb": "ruby",
    "pde": "processing --sketch={{pwd}} --run"
  }
}
```

The above nodemon.json file might be my global config so that I have support for ruby files and processing files, and I can run nodemon demo.pde and nodemon will automatically know how to run the script even though out of the box support for processing scripts.

A further example of options can be seen in sample-nodemon.md

package.json

If you want to keep all your package configurations in one place, nodemon supports using package.json for configuration. Specify the config in the same format as you would for a config file but under nodemonConfig in the package.json file, for example, take the following package.json:

```
{
  "name": "nodemon",
  "homepage": "http://nodemon.io",
  "...": "... other standard package.json values",
  "nodemonConfig": {
      "ignore": ["**/test/**", "**/docs/**"],
      "delay": 2500
  }
}
```

Note that if you specify a --config file or provide a local nodemon.json any package.json config is ignored.

This section needs better documentation, but for now you can also see nodemon --help config (also here).