Reference for various topics

© Prashanth Puranik, www.digdeeper.in

Installation and other tools

Step 1: Install Node - it is installed via a wizard (accept all defaults during installation process). To test if Node is installed correctly, open the command prompt and type the following - each should display the installed version of node and npm CLI tools as output. If the command isn't found, there is an error in the installation.

```
$ node --version
```

```
$ npm --version
```

Step 2: Install nodemon

```
$ npm install -g nodemon
```

Highly recommended: Make sure to use an IDE with support for Node.js - Example Visual Studio Code

Setting environment variables

Mac and Linux

To check environment variables

```
$ env
```

To set an environment variable

```
$ PORT="3000"
$ export PORT
```

References

- http://how-to.wikia.com/wiki/How_to_set,_print,_or_list_environment_variables
- https://stackoverflow.com/questions/7501678/set-environment-variables-on-mac-os-x-lion

Windows

You can set using the Control Panel, the command line (cmd), or PowerShell.

Examples: TBD

References

- https://www.digitalcitizen.life/simple-questions-what-are-environment-variables
- http://www.dowdandassociates.com/blog/content/howto-set-an-environment-variable-in-windows-command-line-and-registry/

Using node REPL

- Type node in the command line to start the node REPL

```
$ node
> console.log( 1 + 2 );
3
undefined
>
```

- Loading a script and executing it

  ```
  > .load script.js
  ```

- Save statements executed in current session onto a script

  ```
  > .save script.js
  ```

- Using a multi-line editor

  ```
  > .editor
  ```

  End the multi-line editing session using Ctrl + D
- Requiring a locally installed module

  ```
  > require( 'some-module' );
  ```

  Note that all built-in node modules are loaded by default in the REPL and need not be required.

Node CLI options

The CLI options have shorthand and longhand forms. Shorthand forms are prefixed with a single hyphen and longhand forms with a double hyphen. Shown below are some options (longhands are not shown except for first one).

- Finding the version of node

  ```
  $ node --version
  ```

  Alternatively,

  ```
  $ node -v
  ```

- Check script for syntax errors and execute if free of errors

  ```
  $ node -c script.js
  ```

- Execute a code snippet inline

  ```
  $ node -p "console.log( 'hello, world' );"
  ```

- Pre-load a module. If multiple modules are to be pre-loaded repeat the option multiple times. Shown below is an example of requiring modules when launching the REPL - same can be done when executing a script. Pre-loaded modules are cached by the node runtime and can save some time when the module is required first by the application. They can also be used to run some script before your application code runs.

  ```
  $ node -r "some-module"
  ```

  Apart from the standard options and arguments, Arbitrary command line arguments can be passed to the node process when launching it. These are available inside the node process as process.argv (an array of strings). Note that the options are not part of the arguments list.

  ```
  $ node -p "process.argv" abc 123 hello
  > [ '/path/to/node', 'abc', '123', 'hello' ]
  ```

The node package manifest file

The package.json file maintains project details and list of saved dependencies. Use npm to create a new node package (a project is also a Node package). You will be taken through a setup wizard. Use the -y option to accept all defaults.

```
npm init
```

The package.json file is generated.

References
Official documentation on package.json

Installing and running nodemon

- Install nodemon globally using npm

  ```
  npm install --global nodemon
  ```

  Alternatively,

  ```
  npm install -g nodemon
  ```

- To run a node application (whose bootstrap script is script.js) using nodemon

  ```
  nodemon node script.js
  ```

  Make a change to your application code and verify that nodemon restarts your application. You are thus spared of restarting your web server manually on every change.

Semantic versioning

A semantically-versioned package has three parts a.b.c - node packages follow semantic versioning and npm can use it along with operators to specify version of packages to be installed.

- a: Major version - a major version upgrade has breaking changes, significant API changes/additions/removal etc.
- b: Minor version - a minor version upgrade has API changes but no breaking changes (backward compatible with previous version)
- c: Patch version - a patch version upgrade is for bug fixes
- Apart from this, a suffix for test versions etc. can exist (eg. -alpha.2)

Using semantic versioning

We use operators to specify package version (or range of versions) in package.json

- =2.1.0 - this is the default (same as "2.1.0")
- <2.1.0 - all versions upto, but not including 2.1.0
- <=2.1.0 - all versions upto, and including 2.1.0
- >=2.1.0 - all versions after, and including 2.1.0
- >=1.2.3 <=2.5.3 - all versions in the specified range
- 5.7, 5.7.x, 5.7.* - any patch of 5.7 version is ok
- ^3, 3.x, 3.x.x - any release with major version 3
- ^3.1.2 - any release after 3.1.2 with major version 3
- ~4.5.3 - same as 4.5.x but x >= 3
- ~4.5 - same as 4.5.x
- ~4 - same as 4.x

References

- Semver specs

- Video explaining semantic versioning in npm/package.json
- Official docs on semantic versioning and operators in npm

NPM - The Node Package Manager

The npm tool manages search, installation, upgrade, uninstallation of node packages, as well as versioning and publishing of npm packages to the node package registries. It can work with the the official npm registry, or a local npm registry (say, set up by a company), or even Git project hosting solutions, eg. GitHub. By default it is configured to work with the official npm registry. You can also have packages searched and installed from your local system!

General npm CLI commands

- The version of npm tool (the longhand displays more details)

  ```
  $ npm --version
  ```

  Alternatively,

  ```
  $ npm -v
  ```

- npm can be updated using npm.

  ```
  $ npm update -g npm
  ```

npm CLI commands and options related to working with a project and using third-party packages

- Creating a node project. You will be taken through a setup wizard. Use the -y option to accept all defaults.

  ```
  $ npm init
  ```

- Searching for a package (in the configured registry)

  ```
  $ npm search some-package
  ```

  Alternatively,

  ```
  $ npm s some-package
  ```

- Installing a package locally (i.e. local to the node project)

  ```
  $ npm install some-package
  ```

  Alternatively,

  ```
  $ npm i some-package
  ```

- Installing all dependencies mentioned in package.json

  ```
  $ npm install
  ```

  Alternatively,

  ```
  $ npm i
  ```

- Saving as a project dependency while installing a package locally. Make sure to use these flags when installing a new package - else a team member will not have these packages installed when running `npm install`

  ```
  $ npm install --save some-package
  ```

For a build time dependency use --save-dev flag

```
$ npm install --save-dev some-package
```

- Installing a package globally. Globally installed packages are stored in a centrally configured folder in your system. These are usually CLI tools written as node packages.

  ```
  $ npm install --global some-package
  ```

  Alternatively,

  ```
  $ npm i -g some-package
  ```

- To update a particular dependency. Use -g (or --global) flag for globally installed modules. up is the shorthand for update.

  ```
  $ npm up some-package
  ```

Exercise: Find out how to uninstall and list installed packages (both global and local). Also explore the depth and output formatting options (like json etc.). Also explore the "home", "repo" commands, and the "outdated" and "prune" commands - these help keep your project dependencies up-to-date and making sure those installed are the necessary ones.

npm CLI commands for configuring how npm behaves in a project

npm behavior can be controlled using npm command line tool options, environment variables and the .npmrc file. The .npmrc files can exist at project (top-level project folder, i.e. along with package.json), user, global and npm level and resolution in case of conflicts is done in a pre-specified order (project config > user config > global config > npm level config). We usually work using either command line tool options, or a project-specific .npmrc file (an ini-formatted list of key = value parameters).

- To list current configuration (after conflict resolution) use

  ```
  npm config list -l
  ```

- To set a config in the global npmrc file using command line tool options

  ```
  npm config set key value
  ```

  Example:

  ```
  npm config set fetch-retries 3
  npm config set save true
  ```

  The above sets number of retries when installing or updating a package from the npm registry, and makes sure all installed packages are saved by default as production dependencies.
- To set the (global) defaults used in `npm init` use

  ```
  npm config set init-author-email "puranik@digdeeper.in"
  npm config set init-license "MIT"
  ```

Exercise: Explore some other config sub-commands (list, delete, edit) and configuration options.

References

- npm CLI documentation
- npmrc
- npm config sub-commands
- npm-config
- DefinitelyTyped website - repo for type definition files
- To search for any TypeScript type definition file

© Prashanth Puranik, www.digdeeper.in