



Full credit is given to the above companies including the Operating System (OS) that this PDF file was generated!

Rocky Enterprise Linux 9.2 Manual Pages on command 'npm-init.1'

\$ man npm-init.1

NPM-INIT(1)

NPM-INIT(1)

NAME

npm-init - Create a package.json file

Synopsis

npm init [--yes|-y|--scope]

npm init <@scope> (same as `npm exec <@scope>/create`)

npm init [<@scope>/]<name> (same as `npm exec [<@scope>/]create-<name>`)

npm init [-w <dir>] [args...]

Description

npm init <initializer> can be used to set up a new or existing npm package.

initializer in this case is an npm package named create-<initializer>, which will be in?

stalled by npm help npm-exec, and then have its main bin executed -- presumably creating

or updating package.json and running any other initialization-related operations.

The init command is transformed to a corresponding npm exec operation as follows:

? npm init foo -> npm exec create-foo

? npm init @usr/foo -> npm exec @usr/create-foo

? npm init @usr -> npm exec @usr/create

If the initializer is omitted (by just calling npm init), init will fall back to legacy

init behavior. It will ask you a bunch of questions, and then write a package.json for

you. It will attempt to make reasonable guesses based on existing fields, dependencies,

and options selected. It is strictly additive, so it will keep any fields and values that

were already set. You can also use -y/--yes to skip the questionnaire altogether. If you

pass --scope, it will create a scoped package.

Forwarding additional options

Any additional options will be passed directly to the command, so `npm init foo -- --hello` will map to `npm exec -- create-foo --hello`.

To better illustrate how options are forwarded, here's a more evolved example showing options passed to both the npm cli and a create package, both following commands are equivalent:

Example 1:

```
? npm init foo -y --registry=<url> -- --hello -a
```

```
? npm exec -y --registry=<url> -- create-foo --hello -a
```

Examples

Create a new React-based project using create-react-app <https://npm.im/create-react-app>:

```
$ npm init react-app ./my-react-app
```

Create a new esm-compatible package using create-esm <https://npm.im/create-esm>:

```
$ mkdir my-esm-lib && cd my-esm-lib
```

```
$ npm init esm --yes
```

Generate a plain old package.json using legacy init:

```
$ mkdir my-npm-pkg && cd my-npm-pkg
```

```
$ git init
```

```
$ npm init
```

Generate it without having it ask any questions:

```
$ npm init -y
```

Workspaces support

It's possible to create a new workspace within your project by using the workspace config option. When using `npm init -w <dir>` the cli will create the folders and boilerplate expected while also adding a reference to your project package.json "workspaces": [] property in order to make sure that new generated workspace is properly set up as such.

Given a project with no workspaces, e.g:

```
.  
+-- package.json
```

You may generate a new workspace using the legacy init:

```
$ npm init -w packages/a
```

That will generate a new folder and package.json file, while also updating your top-level package.json to add the reference to this new workspace:

```
.
```

```

+-- package.json
|-- packages
  |-- a
    |-- package.json

```

The `workspaces init` also supports the `npm init <initializer> -w <dir>` syntax, following the same set of rules explained earlier in the initial Description section of this page. Similar to the previous example of creating a new React-based project using `create-re? act-app` <https://npm.im/create-react-app>, the following syntax will make sure to create the new react app as a nested workspace within your project and configure your `package.json` to recognize it as such:

```
npm init -w packages/my-react-app react-app .
```

This will make sure to generate your react app as expected, one important consideration to have in mind is that `npm exec` is going to be run in the context of the newly created folder for that workspace, and that's the reason why in this example the `initializer` uses the `initializer` name followed with a dot to represent the current directory in that con? text, e.g: `react-app .:`

```

.
+-- package.json
|-- packages
  |-- a
    |-- package.json
  |-- my-react-app
    |-- README
    |-- package.json
  |-- ...

```

Configuration

```

<!-- AUTOGENERATED CONFIG DESCRIPTIONS START --> <!-- automatically generated, do not edit
manually --> <!-- see lib/utis/config/definitions.js -->

```

yes

? Default: null

? Type: null or Boolean

Automatically answer "yes" to any prompts that npm might print on the command line. <!-- automatically generated, do not edit manually --> <!-- see lib/utis/config/definitions.js

-->

force

? Default: false

? Type: Boolean

Removes various protections against unfortunate side effects, common mistakes, unnecessary performance degradation, and malicious input.

? Allow clobbering non-npm files in global installs.

? Allow the npm version command to work on an unclean git repository.

? Allow deleting the cache folder with npm cache clean.

? Allow installing packages that have an engines declaration requiring a different version of npm.

? Allow installing packages that have an engines declaration requiring a different version of node, even if --engine-strict is enabled.

? Allow npm audit fix to install modules outside your stated dependency range (including SemVer-major changes).

? Allow unpublishing all versions of a published package.

? Allow conflicting peerDependencies to be installed in the root project.

? Implicitly set --yes during npm init.

? Allow clobbering existing values in npm pkg

If you don't have a clear idea of what you want to do, it is strongly recommended that you do not use this option! <!-- automatically generated, do not edit manually --> <!-- see lib/utils/config/definitions.js -->

workspace

? Default:

? Type: String (can be set multiple times)

Enable running a command in the context of the configured workspaces of the current project while filtering by running only the workspaces defined by this configuration option.

Valid values for the workspace config are either:

? Workspace names

? Path to a workspace directory

? Path to a parent workspace directory (will result in selecting all workspaces within that folder)

When set for the npm init command, this may be set to the folder of a workspace which does not yet exist, to create the folder and set it up as a brand new workspace within the project.

This value is not exported to the environment for child processes. <!-- automatically generated, do not edit manually --> <!-- see lib/utils/config/definitions.js -->

workspaces

? Default: null

? Type: null or Boolean

Set to true to run the command in the context of all configured workspaces.

Explicitly setting this to false will cause commands like install to ignore workspaces altogether. When not set explicitly:

? Commands that operate on the node_modules tree (install, update, etc.) will link workspaces into the node_modules folder. - Commands that do other things (test, exec, publish, etc.) will operate on the root project, unless one or more workspaces are specified in the workspace config.

This value is not exported to the environment for child processes. <!-- automatically generated, do not edit manually --> <!-- see lib/utils/config/definitions.js -->

include-workspace-root

? Default: false

? Type: Boolean

Include the workspace root when workspaces are enabled for a command.

When false, specifying individual workspaces via the workspace config, or all workspaces via the workspaces flag, will cause npm to operate only on the specified workspaces, and not on the root project. <!-- automatically generated, do not edit manually --> <!-- see lib/utils/config/definitions.js -->

<!-- AUTOGENERATED CONFIG DESCRIPTIONS END -->

See Also

? init-package-json module <http://npm.im/init-package-json>

? npm help package.json

? npm help version

? npm help scope

? npm help exec

? npm help workspaces

undefined NaN

NPM-INIT(1)