MyWebUniversity
python
Red Hat Enterprise Linux Manual Pages
FPDF Library PDF generator
FPDF

Red Hat Enterprise Linux Release 9.2 Manual Pages on 'workspaces.7' command

$ man workspaces.7

WORKSPACES(7) WORKSPACES(7)

NAME

workspaces - Working with workspaces

Description

Workspaces is a generic term that refers to the set of features in the
npm cli that provides support to managing multiple packages from your
local file system from within a singular top-level, root package.

This set of features makes up for a much more streamlined workflow han?
dling linked packages from the local file system. Automating the link?
ing process as part of npm install and avoiding manually having to use
npm link in order to add references to packages that should be sym?
linked into the current node_modules folder.

We also refer to these packages being auto-symlinked during npm install
as a single workspace, meaning it's a nested package within the current
local file system that is explicitly defined in the package.json /con?
figuring-npm/package-json#workspaces workspaces configuration.

Defining workspaces

Workspaces are usually defined via the workspaces property of the pack?
age.json /configuring-npm/package-json#workspaces file, e.g:

```
{
  "name": "my-workspaces-powered-project",
  "workspaces": [
    "packages/a"
```

```
     ]
   }
```

Given the above package.json example living at a current working direc?
tory . that contains a folder named packages/a that itself contains a
package.json inside it, defining a Node.js package, e.g:

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

The expected result once running npm install in this current working
directory . is that the folder packages/a will get symlinked to the
node_modules folder of the current working dir.

Below is a post npm install example, given that same previous example
structure of files and folders:

```
  .
  +-- node_modules
  |  `-- a -> ../packages/a
  +-- package-lock.json
  +-- package.json
  `-- packages
     +-- a
     |   `-- package.json
```

Getting started with workspaces

You may automate the required steps to define a new workspace using npm
help init. For example in a project that already has a package.json de?
fined you can run:

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

This command will create the missing folders and a new package.json
file (if needed) while also making sure to properly configure the
"workspaces" property of your root project package.json.

Adding dependencies to a workspace

It's possible to directly add/remove/update dependencies of your workspaces using the workspace config /using-npm/config#workspace.

For example, assuming the following structure:

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

If you want to add a dependency named abbrev from the registry as a de?
pendency of your workspace a, you may use the workspace config to tell
the npm installer that package should be added as a dependency of the
provided workspace:

```
npm install abbrev -w a
```

Note: other installing commands such as uninstall, ci, etc will also
respect the provided workspace configuration.

Using workspaces

Given the specifities of how Node.js handles module resolution
https://nodejs.org/dist/latest-v14.x/docs/api/modules.html#mod?
ules_all_together it's possible to consume any defined workspace by its
declared package.json name. Continuing from the example defined above,
let's also create a Node.js script that will require the workspace a
example module, e.g:

```
// ./packages/a/index.js
module.exports = 'a'
// ./lib/index.js
const moduleA = require('a')
console.log(moduleA) // -> a
```

When running it with:

```
node lib/index.js
```

This demonstrates how the nature of node_modules resolution allows for
workspaces to enable a portable workflow for requiring each workspace
in such a way that is also easy to npm help publish these nested

workspaces to be consumed elsewhere.

Running commands in the context of workspaces

You can use the workspace configuration option to run commands in the context of a configured workspace. Additionally, if your current di?
rectory is in a workspace, the workspace configuration is implicitly set, and prefix is set to the root workspace.

Following is a quick example on how to use the npm run command in the context of nested workspaces. For a project containing multiple workspaces, e.g:

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

By running a command using the workspace option, it's possible to run the given command in the context of that specific workspace. e.g:

```
npm run test --workspace=a
```

You could also run the command within the workspace.

```
cd packages/a && npm run test
```

Either will run the test script defined within the ./packages/a/pack?
age.json file.

Please note that you can also specify this argument multiple times in the command-line in order to target multiple workspaces, e.g:

```
npm run test --workspace=a --workspace=b
```

It's also possible to use the workspaces (plural) configuration option to enable the same behavior but running that command in the context of all configured workspaces. e.g:

```
npm run test --workspaces
```

Will run the test script in both ./packages/a and ./packages/b.

Commands will be run in each workspace in the order they appear in your package.json

```
  {
    "workspaces": [ "packages/a", "packages/b" ]
  }
```

Order of run is different with:

```
  {
    "workspaces": [ "packages/b", "packages/a" ]
  }
```

Ignoring missing scripts

It is not required for all of the workspaces to implement scripts run with the npm run command.

By running the command with the --if-present flag, npm will ignore workspaces missing target script.

```
npm run test --workspaces --if-present
```

See also

? npm help install

? npm help publish

? npm help run-script

? npm help config