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 handling linked packages from the local file system. Automating the linking process as part of `npm install` and avoiding manually having to use `npm link` in order to add references to packages that should be symlinked 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` `workspaces` configuration.

Defining workspaces

Workspaces are usually defined via the `workspaces` property of the `package.json` file, e.g:

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

Given the above `package.json` example living at a current working directory `.` 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
|  `-- packages/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 init. For example in a project that already has a `package.json` defined 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.

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 dependency 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 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:

```
// ./workspace-a/index.js
module.exports = 'a'

// ./lib/index.js
const moduleA = require('workspace-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 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.

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
```

This will run the `test` script defined within the `./packages/a/package.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 install

- npm publish
- npm run-script
- config