npm-link

Symlink a package folder

SYNOPSIS

```
npm link (in package dir)
npm link [<@scope>/]<pkg>[@<version>]

alias: npm ln
```

DESCRIPTION

Package linking is a two-step process.

First, `npm link` in a package folder will create a symlink in the global folder `{prefix}/lib/node_modules/<package>` that links to the package where the `npm link` command was executed. (see `npm-config` for the value of `prefix`). It will also link any bins in the package to `{prefix}/bin/{name}` .

Next, in some other location, `npm link package-name` will create a symbolic link from globally-installed `package-name` to `node_modules/` of the current folder.

Note that `package-name` is taken from `package.json` , not from directory name.

The package name can be optionally prefixed with a scope. See `npm-scope` . The scope must be preceded by an @-symbol and followed by a slash.

When creating tarballs for `npm publish` , the linked packages are “snapshotted” to their current state by resolving the symbolic links.

This is handy for installing your own stuff, so that you can work on it and test it iteratively without having to continually rebuild.

For example:

```
cd ~/projects/node-redis    # go into the package directory
npm link                    # creates global link
cd ~/projects/node-bloggy   # go into some other package directory.
npm link redis              # link-install the package
```

Now, any changes to ~/projects/node-redis will be reflected in ~/projects/node-bloggy/node_modules/node-redis/. Note that the link should be to the package name, not the directory name for that package.

You may also shortcut the two steps in one. For example, to do the above use-case in a shorter way:

```
cd ~/projects/node-bloggy  # go into the dir of your main project
npm link ../node-redis     # link the dir of your dependency
```

The second line is the equivalent of doing:

```
(cd ../node-redis; npm link)
npm link redis
```

That is, it first creates a global link, and then links the global installation target into your project's `node_modules` folder.

Note that in this case, you are referring to the directory name, `node-redis`, rather than the package name `redis`.

If your linked package is scoped (see `npm-scope`) your link command must include that scope, e.g.

```
npm link @myorg/privatepackage
```