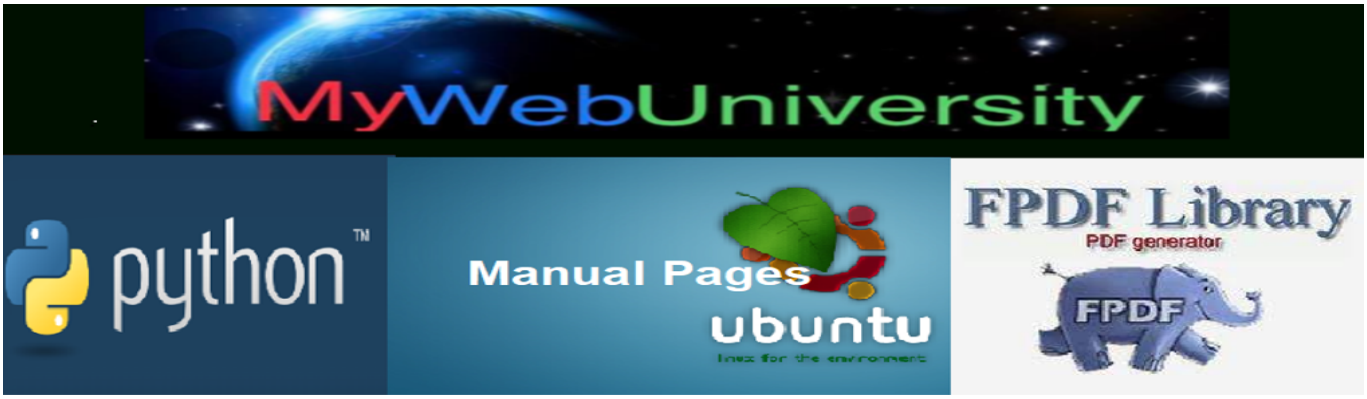

Rocky Enterprise Linux 9.2 Manual Pages on command 'npm-update.1'

$ man npm-update.1

NPM-UPDATE(1) NPM-UPDATE(1)

NAME

npm-update - Update packages

Synopsis

npm update [-g] [<pkg>...]

aliases: up, upgrade

Description

This command will update all the packages listed to the latest version (specified by the tag config), respecting the semver constraints of both your package and its dependencies (if they also require the same package).

It will also install missing packages.

If the -g flag is specified, this command will update globally installed packages.

If no package name is specified, all packages in the specified location (global or local) will be updated.

Note that by default npm update will not update the semver values of direct dependencies in your project package.json, if you want to also update values in package.json you can run: npm update --save (or add the save=true option to a npm help configuration file to make that the default behavior).

Example

For the examples below, assume that the current package is app and it depends on dependen? cies, dep1 (dep2, .. etc.). The published versions of dep1 are:

{

"dist-tags": { "latest": "1.2.2" },

```
    "versions": [
      "1.2.2",
      "1.2.1",
      "1.2.0",
      "1.1.2",
      "1.1.1",
      "1.0.0",
      "0.4.1",
      "0.4.0",
      "0.2.0"
    ]
  }
```

Caret Dependencies

If app's package.json contains:

```
  "dependencies": {
    "dep1": "^1.1.1"
  }
```

Then npm update will install dep1@1.2.2, because 1.2.2 is latest and 1.2.2 satisfies ^1.1.1.

Tilde Dependencies

However, if app's package.json contains:

```
  "dependencies": {
    "dep1": "~1.1.1"
  }
```

In this case, running npm update will install dep1@1.1.2. Even though the latest tag points to 1.2.2, this version do not satisfy ~1.1.1, which is equivalent to >=1.1.1 <1.2.0. So the highest-sorting version that satisfies ~1.1.1 is used, which is 1.1.2.

Caret Dependencies below 1.0.0

Suppose app has a caret dependency on a version below 1.0.0, for example:

```
  "dependencies": {
    "dep1": "^0.2.0"
  }
```

npm update will install dep1@0.2.0, because there are no other versions which satisfy

^0.2.0.

If the dependence were on ^0.4.0:

```
"dependencies": {
  "dep1": "^0.4.0"
}
```

Then npm update will install dep1@0.4.1, because that is the highest-sorting version that satisfies ^0.4.0 (>= 0.4.0 <0.5.0)

Subdependencies

Suppose your app now also has a dependency on dep2

```
{
  "name": "my-app",
  "dependencies": {
      "dep1": "^1.0.0",
      "dep2": "1.0.0"
  }
}
```

and dep2 itself depends on this limited range of dep1

```
{
 "name": "dep2",
  "dependencies": {
    "dep1": "~1.1.1"
  }
}
```

Then npm update will install dep1@1.1.2 because that is the highest version that dep2 allows. npm will prioritize having a single version of dep1 in your tree rather than two when that single version can satisfy the semver requirements of multiple dependencies in your tree. In this case if you really did need your package to use a newer version you would need to use npm install.

Updating Globally-Installed Packages

npm update -g will apply the update action to each globally installed package that is outdated -- that is, has a version that is different from wanted.

Note: Globally installed packages are treated as if they are installed with a caret semver range specified. So if you require to update to latest you may need to run npm install -g

[<pkg>...]

NOTE: If a package has been upgraded to a version newer than latest, it will be down?

graded.

Configuration

<!-- AUTOGENERATED CONFIG DESCRIPTIONS START --> <!-- automatically generated, do not edit

manually --> <!-- see lib/utils/config/definitions.js -->

global

? Default: false

? Type: Boolean

Operates in "global" mode, so that packages are installed into the prefix folder instead

of the current working directory. See npm help folders for more on the differences in be?

havior.

? packages are installed into the {prefix}/lib/node_modules folder, instead of the current

working directory.

? bin files are linked to {prefix}/bin

? man pages are linked to {prefix}/share/man

<!-- automatically generated, do not edit manually --> <!-- see lib/utils/config/defini?

tions.js -->

global-style

? Default: false

? Type: Boolean

Causes npm to install the package into your local node_modules folder with the same layout

it uses with the global node_modules folder. Only your direct dependencies will show in

node_modules and everything they depend on will be flattened in their node_modules fold?

ers. This obviously will eliminate some deduping. If used with legacy-bundling,

legacy-bundling will be preferred. <!-- automatically generated, do not edit manually -->

<!-- see lib/utils/config/definitions.js -->

legacy-bundling

? Default: false

? Type: Boolean

Causes npm to install the package such that versions of npm prior to 1.4, such as the one

included with node 0.8, can install the package. This eliminates all automatic deduping.

If used with global-style this option will be preferred. <!-- automatically generated, do

not edit manually --> <!-- see lib/utils/config/definitions.js -->

strict-peer-deps

? Default: false

? Type: Boolean

If set to true, and --legacy-peer-deps is not set, then any conflicting peerDependencies will be treated as an install failure, even if npm could reasonably guess the appropriate resolution based on non-peer dependency relationships.

By default, conflicting peerDependencies deep in the dependency graph will be resolved us? ing the nearest non-peer dependency specification, even if doing so will result in some packages receiving a peer dependency outside the range set in their package's peerDepen? dencies object.

When such and override is performed, a warning is printed, explaining the conflict and the packages involved. If --strict-peer-deps is set, then this warning is treated as a fail? ure. <!-- automatically generated, do not edit manually --> <!-- see lib/utils/con? fig/definitions.js -->

package-lock

? Default: true

? Type: Boolean

If set to false, then ignore package-lock.json files when installing. This will also pre? vent writing package-lock.json if save is true.

When package package-locks are disabled, automatic pruning of extraneous modules will also be disabled. To remove extraneous modules with package-locks disabled use npm prune. <!-- automatically generated, do not edit manually --> <!-- see lib/utils/config/definitions.js -->

omit

? Default: 'dev' if the NODE_ENV environment variable is set to 'production', otherwise empty.

? Type: "dev", "optional", or "peer" (can be set multiple times)

Dependency types to omit from the installation tree on disk.

Note that these dependencies are still resolved and added to the package-lock.json or npm-shrinkwrap.json file. They are just not physically installed on disk.

If a package type appears in both the --include and --omit lists, then it will be in? cluded.

If the resulting omit list includes 'dev', then the NODE_ENV environment variable will be set to 'production' for all lifecycle scripts. <!-- automatically generated, do not edit manually --> <!-- see lib/utils/config/definitions.js -->

ignore-scripts

? Default: false

? Type: Boolean

If true, npm does not run scripts specified in package.json files.

Note that commands explicitly intended to run a particular script, such as npm start, npm stop, npm restart, npm test, and npm run-script will still run their intended script if ignore-scripts is set, but they will not run any pre- or post-scripts. <!-- automatically generated, do not edit manually --> <!-- see lib/utils/config/definitions.js -->

audit

? Default: true

? Type: Boolean

When "true" submit audit reports alongside the current npm command to the default registry and all registries configured for scopes. See the documentation for npm help audit for de? tails on what is submitted. <!-- automatically generated, do not edit manually --> <!-- see lib/utils/config/definitions.js -->

bin-links

? Default: true

? Type: Boolean

Tells npm to create symlinks (or .cmd shims on Windows) for package executables.

Set to false to have it not do this. This can be used to work around the fact that some file systems don't support symlinks, even on ostensibly Unix systems. <!-- automatically generated, do not edit manually --> <!-- see lib/utils/config/definitions.js -->

fund

? Default: true

? Type: Boolean

When "true" displays the message at the end of each npm install acknowledging the number of dependencies looking for funding. See npm help npm fund for details. <!-- automati? cally generated, do not edit manually --> <!-- see lib/utils/config/definitions.js -->

dry-run

? Default: false

? Type: Boolean

Indicates that you don't want npm to make any changes and that it should only report what it would have done. This can be passed into any of the commands that modify your local in? stallation, eg, install, update, dedupe, uninstall, as well as pack and publish.

Note: This is NOT honored by other network related commands, eg dist-tags, owner, etc.

<!-- automatically generated, do not edit manually --> <!-- see lib/utils/config/defini? tions.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 op? tion.

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 al? together. 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 spec? ified 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

? npm help install

? npm help outdated

? npm help shrinkwrap

? npm help registry

? npm help folders

? npm help ls