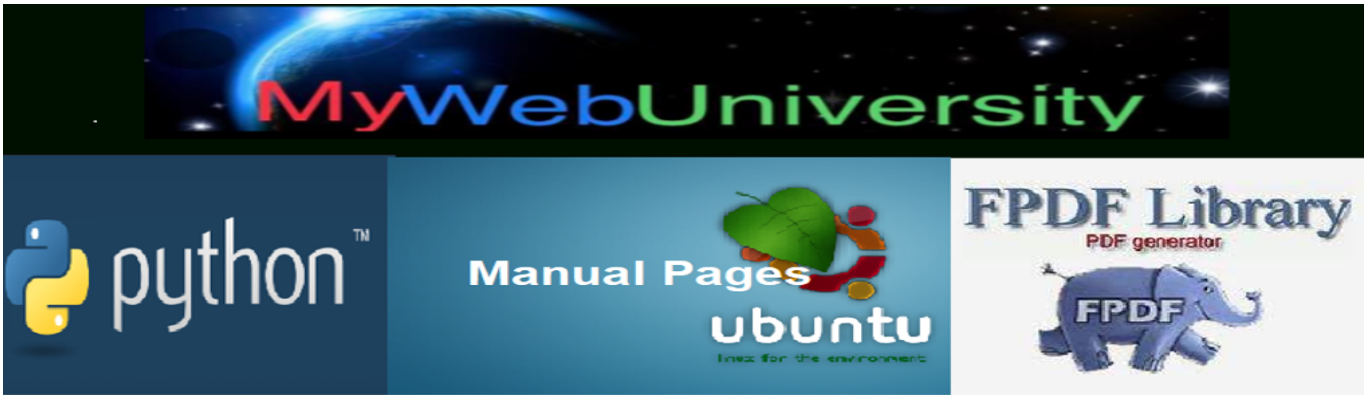

Full credit is given to the above companies including the Operating System (OS) that this PDF file was generated!

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

$ man npm-install.1

NPM-INSTALL(1) NPM-INSTALL(1)

NAME

npm-install - Install a package

Synopsis

npm install (with no args, in package dir)

npm install [<@scope>/]<name>

npm install [<@scope>/]<name>@<tag>

npm install [<@scope>/]<name>@<version>

npm install [<@scope>/]<name>@<version range>

npm install <alias>@npm:<name>

npm install <git-host>:<git-user>/<repo-name>

npm install <git repo url>

npm install <tarball file>

npm install <tarball url>

npm install <folder>

aliases: npm i, npm add

common options: [-P|--save-prod|-D|--save-dev|-O|--save-optional|--save-peer] [-E|--save-exact] [-B|--save-bundle] [--no-save] [--dry-run]

Description

This command installs a package and any packages that it depends on. If the package has a package-lock, or an npm shrinkwrap file, or a yarn lock file, the installation of depen? dencies will be driven by that, respecting the following order of precedence:

? npm-shrinkwrap.json

? package-lock.json

? yarn.lock

See npm help package-lock.json and npm help shrinkwrap.

A package is:

? a) a folder containing a program described by a npm help package.json file

? b) a gzipped tarball containing (a)

? c) a url that resolves to (b)

? d) a <name>@<version> that is published on the registry (see npm help registry) with (c)

? e) a <name>@<tag> (see npm help dist-tag) that points to (d)

? f) a <name> that has a "latest" tag satisfying (e)

? g) a <git remote url> that resolves to (a)

Even if you never publish your package, you can still get a lot of benefits of using npm if you just want to write a node program (a), and perhaps if you also want to be able to easily install it elsewhere after packing it up into a tarball (b).

? npm install (in a package directory, no arguments):

Install the dependencies to the local node_modules folder.

In global mode (ie, with -g or --global appended to the command), it installs the current package context (ie, the current working directory) as a global package.

By default, npm install will install all modules listed as dependencies in npm help package.json.

With the --production flag (or when the NODE_ENV environment variable is set to production), npm will not install modules listed in devDependencies. To install all modules listed in both dependencies and devDependencies when NODE_ENV environment variable is set to production, you can use --production=false. NOTE: The --production flag has no particular meaning when adding a dependency to a project.

? npm install <folder>:

If <folder> sits inside the root of your project, its dependencies will be installed and may be hoisted to the top-level node_modules as they would for other types of dependencies. If <folder> sits outside the root of your project,

npm will not install the package dependencies in the directory <folder>,

but it will create a symlink to <folder>. NOTE: If you want to install the content of a directory like a package from the registry instead of creating a link, you would need to use npm help pack while in the <folder> directory, and then install the resulting tarball instead of the <folder> using npm install <tarball file>

Example:

npm install ../../other-package

npm install ./sub-package

? npm install <tarball file>:

Install a package that is sitting on the filesystem. Note: if you just want to link a dev directory into your npm root, you can do this more easily by using npm help link.

Tarball requirements:

? The filename must use .tar, .tar.gz, or .tgz as the extension.

? The package contents should reside in a subfolder inside the tarball (usually it is called package/). npm strips one directory layer when installing the package (an equivalent of tar x --strip-components=1 is run).

? The package must contain a package.json file with name and version properties. Exam? ple:

npm install ./package.tgz

? npm install <tarball url>:

Fetch the tarball url, and then install it. In order to distinguish between this and other options, the argument must start with "http://" or "https://"

Example:

npm install https://github.com/indexzero/forever/tarball/v0.5.6

? npm install [<@scope>/]<name>:

Do a <name>@<tag> install, where <tag> is the "tag" config. (See npm help config. The config's default value is latest.)

In most cases, this will install the version of the modules tagged as latest on the npm registry.

Example:

npm install sax

npm install saves any specified packages into dependencies by default.

Additionally, you can control where and how they get saved with some additional flags:

? -P, --save-prod: Package will appear in your dependencies. This is the default unless -D or -O are present.

? -D, --save-dev: Package will appear in your devDependencies.

? -O, --save-optional: Package will appear in your optionalDependencies.

? --no-save: Prevents saving to dependencies. When using any of the above options to save dependencies to your package.json, there are two additional, optional flags:

? -E, --save-exact: Saved dependencies will be configured with an exact version rather than using npm's default semver range operator.

? -B, --save-bundle: Saved dependencies will also be added to your bundleDependen? cies list. Further, if you have an npm-shrinkwrap.json or package-lock.json then it will be updated as well. <scope> is optional. The package will be downloaded from the registry associated with the specified scope. If no registry is associ? ated with the given scope the default registry is assumed. See npm help scope. Note: if you do not include the @-symbol on your scope name, npm will interpret this as a GitHub repository instead, see below. Scopes names must also be fol? lowed by a slash. Examples:

npm install sax

npm install githubname/reponame

npm install @myorg/privatepackage

npm install node-tap --save-dev

npm install dtrace-provider --save-optional

npm install readable-stream --save-exact

npm install ansi-regex --save-bundle

? Note*: If there is a file or folder named <name> in the current working direc? tory, then it will try to install that, and only try to fetch the package by name if it is not valid.

? npm install <alias>@npm:<name>:

Install a package under a custom alias. Allows multiple versions of a same-name package side-by-side, more convenient import names for packages with otherwise long ones, and using git forks replacements

or forked npm packages as replacements. Aliasing works only on your

project and does not rename packages in transitive dependencies.

Aliases should follow the naming conventions stated in

validate-npm-package-name https://www.npmjs.com/package/validate-npm-package-name#nam?

ing-rules.

Examples:

npm install my-react@npm:react

npm install jquery2@npm:jquery@2

npm install jquery3@npm:jquery@3

npm install npa@npm:npm-package-arg

? npm install [<@scope>/]<name>@<tag>:

Install the version of the package that is referenced by the specified tag.

If the tag does not exist in the registry data for that package, then this

will fail.

Example:

npm install sax@latest

npm install @myorg/mypackage@latest

? npm install [<@scope>/]<name>@<version>:

Install the specified version of the package. This will fail if the

version has not been published to the registry.

Example:

npm install sax@0.1.1

npm install @myorg/privatepackage@1.5.0

? npm install [<@scope>/]<name>@<version range>:

Install a version of the package matching the specified version range.

This will follow the same rules for resolving dependencies described in

npm help package.json.

Note that most version ranges must be put in quotes so that your shell

will treat it as a single argument.

Example:

npm install sax@">=0.1.0 <0.2.0"

npm install @myorg/privatepackage@"16 - 17"

? npm install <git remote url>:

Installs the package from the hosted git provider, cloning it with

git. For a full git remote url, only that URL will be attempted.

<protocol>://[<user>[:<password>]@]<hostname>[:<port>][:][/]<path>[#<commit-ish> | #semver:<semver>]

<protocol> is one of git, git+ssh, git+http, git+https, or

git+file.

If #<commit-ish> is provided, it will be used to clone exactly that

commit. If the commit-ish has the format #semver:<semver>, <semver>

can be any valid semver range or exact version, and npm will look for

any tags or refs matching that range in the remote repository, much as

it would for a registry dependency. If neither #<commit-ish> or

#semver:<semver> is specified, then the default branch of the

repository is used.

If the repository makes use of submodules, those submodules will be

cloned as well.

If the package being installed contains a prepare script, its

dependencies and devDependencies will be installed, and the prepare

script will be run, before the package is packaged and installed.

The following git environment variables are recognized by npm and will

be added to the environment when running git:

? GIT_ASKPASS

? GIT_EXEC_PATH

? GIT_PROXY_COMMAND

? GIT_SSH

? GIT_SSH_COMMAND

? GIT_SSL_CAINFO

? GIT_SSL_NO_VERIFY See the git man page for details. Examples:

npm install git+ssh://git@github.com:npm/cli.git#v1.0.27

npm install git+ssh://git@github.com:npm/cli#pull/273

npm install git+ssh://git@github.com:npm/cli#semver:^5.0

npm install git+https://isaacs@github.com/npm/cli.git

npm install git://github.com/npm/cli.git#v1.0.27

GIT_SSH_COMMAND='ssh -i ~/.ssh/custom_ident' npm install git+ssh://git@github.com:npm/cli.git

? npm install <githubname>/<githubrepo>[#<commit-ish>]:

? npm install github:<githubname>/<githubrepo>[#<commit-ish>]:

Install the package at https://github.com/githubname/githubrepo by attempting to clone it using git.

If #<commit-ish> is provided, it will be used to clone exactly that commit. If the commit-ish has the format #semver:<semver>, <semver> can be any valid semver range or exact version, and npm will look for any tags or refs matching that range in the remote repository, much as it would for a registry dependency. If neither #<commit-ish> or #semver:<semver> is specified, then master is used.

As with regular git dependencies, dependencies and devDependencies will be installed if the package has a prepare script before the package is done installing.

Examples:

npm install mygithubuser/myproject

npm install github:mygithubuser/myproject

? npm install gist:[<githubname>/]<gistID>[#<commit-ish>|#semver:<semver>]:

Install the package at https://gist.github.com/gistID by attempting to clone it using git. The GitHub username associated with the gist is optional and will not be saved in package.json.

As with regular git dependencies, dependencies and devDependencies will be installed if the package has a prepare script before the package is done installing.

Example:

npm install gist:101a11beef

? npm install bitbucket:<bitbucketname>/<bitbucketrepo>[#<commit-ish>]:

Install the package at https://bitbucket.org/bitbucketname/bitbucketrepo by attempting to clone it using git.

If #<commit-ish> is provided, it will be used to clone exactly that commit. If the commit-ish has the format #semver:<semver>, <semver> can be any valid semver range or exact version, and npm will look for any tags or refs matching that range in the remote repository, much as it would for a registry dependency. If neither #<commit-ish> or #semver:<semver> is specified, then master is used.

As with regular git dependencies, dependencies and devDependencies will
be installed if the package has a prepare script before the package is
done installing.
Example:
npm install bitbucket:mybitbucketuser/myproject

? npm install gitlab:<gitlabname>/<gitlabrepo>[#<commit-ish>]:
Install the package at https://gitlab.com/gitlabname/gitlabrepo
by attempting to clone it using git.
If #<commit-ish> is provided, it will be used to clone exactly that
commit. If the commit-ish has the format #semver:<semver>, <semver> can
be any valid semver range or exact version, and npm will look for any tags
or refs matching that range in the remote repository, much as it would for a
registry dependency. If neither #<commit-ish> or #semver:<semver> is
specified, then master is used.
As with regular git dependencies, dependencies and devDependencies will
be installed if the package has a prepare script before the package is
done installing.
Example:
npm install gitlab:mygitlabuser/myproject
npm install gitlab:myusr/myproj#semver:^5.0

You may combine multiple arguments and even multiple types of arguments. For example:
npm install sax@">=0.1.0 <0.2.0" bench supervisor

The --tag argument will apply to all of the specified install targets. If a tag with the
given name exists, the tagged version is preferred over newer versions.

The --dry-run argument will report in the usual way what the install would have done with?
out actually installing anything.

The --package-lock-only argument will only update the package-lock.json, instead of check?
ing node_modules and downloading dependencies.

The -f or --force argument will force npm to fetch remote resources even if a local copy
exists on disk.
npm install sax --force

Configuration

See the npm help config help doc. Many of the configuration params have some effect on

installation, since that's most of what npm does.

These are some of the most common options related to installation. <!-- AUTOGENERATED CONFIG DESCRIPTIONS START --> <!-- automatically generated, do not edit manually --> <!-- see lib/utils/config/definitions.js -->

save

? Default: true

? Type: Boolean

Save installed packages to a package.json file as dependencies.

When used with the npm rm command, removes the dependency from package.json. <!-- auto? matically generated, do not edit manually --> <!-- see lib/utils/config/definitions.js -->

save-exact

? Default: false

? Type: Boolean

Dependencies saved to package.json will be configured with an exact version rather than using npm's default semver range operator. <!-- automatically generated, do not edit man? ually --> <!-- 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 included.

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 details 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 -->

Algorithm

Given a package{dep} structure: A{B,C}, B{C}, C{D}, the npm install algorithm produces:

A

+-- B

+-- C

+-- D

That is, the dependency from B to C is satisfied by the fact that A already caused C to be installed at a higher level. D is still installed at the top level because nothing con? flicts with it.

For A{B,C}, B{C,D@1}, C{D@2}, this algorithm produces:

A

+-- B

```
+-- C
`-- D@2
+-- D@1
```

Because B's D@1 will be installed in the top-level, C now has to install D@2 privately for itself. This algorithm is deterministic, but different trees may be produced if two depen? dencies are requested for installation in a different order.

See npm help folders for a more detailed description of the specific folder structures that npm creates.

See Also

- ? npm help folders
- ? npm help update
- ? npm help audit
- ? npm help fund
- ? npm help link
- ? npm help rebuild
- ? npm help scripts
- ? npm help config
- ? npm help npmrc
- ? npm help registry
- ? npm help dist-tag
- ? npm help uninstall
- ? npm help shrinkwrap
- ? npm help package.json
- ? npm help workspaces