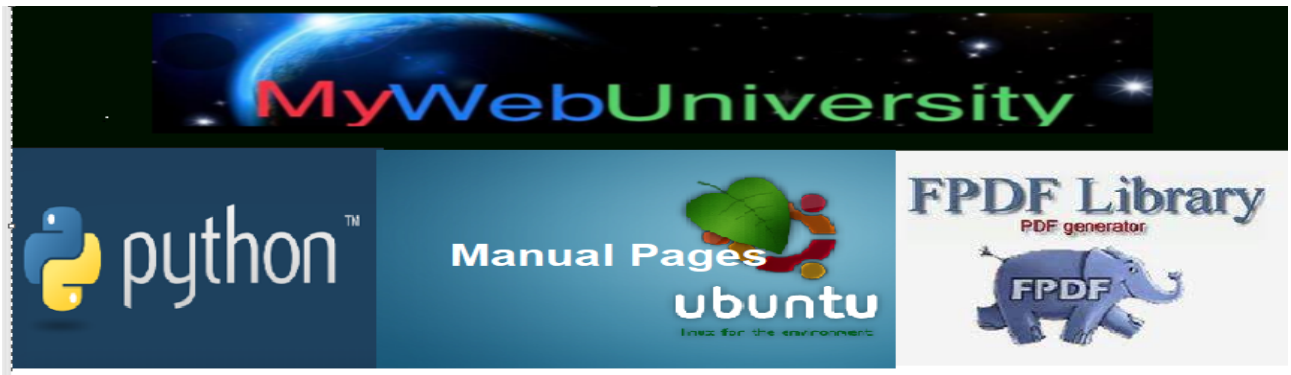

Full credit is given to the above companies including the OS that this PDF file was generated!

Linux Ubuntu 26.04 Manual Pages on command 'npm-install.1'

$ man npm-install.1

NPM-INSTALL(1) General Commands Manual NPM-INSTALL(1)

NAME

npm-install

Synopsis

<!-- AUTOGENERATED USAGE DESCRIPTIONS -->

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 dependencies will be driven by that, respecting the following order of precedence:

? npm-shrinkwrap.json

? package-lock.json

? yarn.lock

See package-lock.json and npm shrinkwrap.

A package is:

? a) a folder containing a program described by a 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 registry) with (c)

? e) a <name>@<tag> (see npm 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 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 the --install-links op? tion.

Example:

npm install ../../other-package --install-links

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 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.

Example:

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 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
bundleDependencies 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 associated with
the given scope the default registry is assumed. See
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 followed 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 directory, 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.
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
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 the default branch 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 without actually installing anything.

The --package-lock-only argument will only update the package-lock.json, instead of checking 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 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 CON?

FIG DESCRIPTIONS -->

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 conflicts 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 dependencies are requested for installation in

a different order.

See folders for a more detailed description of

the specific folder structures that npm creates.

See Also

? npm folders

? npm update

? npm audit

? npm fund

? npm link

? npm rebuild

? npm scripts

? npm config

? npmrc

? npm registry

? npm dist-tag

? npm uninstall

? npm shrinkwrap

? package.json

? workspaces