NAME

npm-version − Bump a package version

Synopsis

```
npm version [<newversion> | major | minor | patch | premajor | preminor | prepatch | prerelease [−−preid=<prerelease−i

 'npm [−v | −−version]' to print npm version
 'npm view <pkg> version' to view a package's published version
 'npm ls' to inspect current package/dependency versions
```

Configuration

<!−− AUTOGENERATED CONFIG DESCRIPTIONS START −−> <!−− automatically generated, do not edit manually −−> <!−− see lib/utils/config/definitions.js −−>

allow−same−version

- Default: false
- Type: Boolean

Prevents throwing an error when **npm version** is used to set the new version to the same value as the current version. <!−− automatically generated, do not edit manually −−> <!−− see lib/utils/config/definitions.js −−>

commit−hooks

- Default: true
- Type: Boolean

Run git commit hooks when using the **npm version** command. <!−− automatically generated, do not edit manually −−> <!−− see lib/utils/config/definitions.js −−>

git−tag−version

- Default: true
- Type: Boolean

Tag the commit when using the **npm version** command. <!−− automatically generated, do not edit manually −−> <!−− see lib/utils/config/definitions.js −−>

json

- Default: false
- Type: Boolean

Whether or not to output JSON data, rather than the normal output.

- In **npm pkg set** it enables parsing set values with JSON.parse() before saving them to your **package.json**.

Not supported by all npm commands. <!−− automatically generated, do not edit manually −−> <!−− see lib/utils/config/definitions.js −−>

preid

- Default: ""

- Type: String

The "prerelease identifier" to use as a prefix for the "prerelease" part of a semver. Like the **rc** in **1.2.0−rc.8** . <!−− automatically generated, do not edit manually −−> <!−− see lib/utils/config/definitions.js −−>

sign−git−tag

- Default: false
- Type: Boolean

If set to true, then the **npm version** command will tag the version using **−s** to add a signature.

Note that git requires you to have set up GPG keys in your git configs for this to work properly. <!−− automatically generated, do not edit manually −−> <!−− see lib/utils/config/definitions.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 option.

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 altogether. 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 specified 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 −−>

Description

Run this in a package directory to bump the version and write the new data back to **package.json**, **package−lock.json**, and, if present, **npm−shrinkwrap.json**.

The **newversion** argument should be a valid semver string, a valid second argument to semver.inc *https://github.com/npm/node−semver#functions* (one of **patch**, **minor**, **major**, **prepatch**, **preminor**, **premajor**, **prerelease**), or **from−git**. In the second case, the existing version will be incremented by 1 in the specified field. **from−git** will try to read the latest git tag, and use that as the new npm version.

If run in a git repo, it will also create a version commit and tag. This behavior is controlled by **git−tag−version** (see below), and can be disabled on the command line by running **npm −−no−git−tag−version version**. It will fail if the working directory is not clean, unless the **−f** or **−−force** flag is set.

If supplied with **−m** or **−−message** config option, npm will use it as a commit message when creating a version commit. If the **message** config contains **%s** then that will be replaced with the resulting version number. For example:

```
npm version patch −m "Upgrade to %s for reasons"
```

If the **sign−git−tag** config is set, then the tag will be signed using the **−s** flag to git. Note that you must have a default GPG key set up in your git config for this to work properly. For example:

```
$ npm config set sign−git−tag true
$ npm version patch

You need a passphrase to unlock the secret key for
user: "isaacs (http://blog.izs.me/) <i@izs.me>"
2048−bit RSA key, ID 6C481CF6, created 2010−08−31

Enter passphrase:
```

If **preversion**, **version**, or **postversion** are in the **scripts** property of the package.json, they will be executed as part of running **npm version**.

The exact order of execution is as follows:

1. Check to make sure the git working directory is clean before we get started. Your scripts may add files to the commit in future steps. This step is skipped if the **−−force** flag is set.
2. Run the **preversion** script. These scripts have access to the old **version** in package.json. A typical use would be running your full test suite before deploying. Any files you want added to the commit should be explicitly added using **git add**.
3. Bump **version** in **package.json** as requested (**patch**, **minor**, **major**, etc).
4. Run the **version** script. These scripts have access to the new **version** in package.json (so they can incorporate it into file headers in generated files for example). Again, scripts should explicitly add generated files to the commit using **git add**.
5. Commit and tag.

6. Run the **postversion** script. Use it to clean up the file system or automatically push the commit and/or tag.

Take the following example:

```
{
 "scripts": {
  "preversion": "npm test",
  "version": "npm run build && git add −A dist",
  "postversion": "git push && git push −−tags && rm −rf build/temp"
 }
}
```

This runs all your tests and proceeds only if they pass. Then runs your **build** script, and adds everything in the **dist** directory to the commit. After the commit, it pushes the new commit and tag up to the server, and deletes the **build/temp** directory.

See Also

- npm help init
- npm help run−script
- npm help scripts
- npm help package.json
- npm help config