Synopsis

```
npm outdated [[<@scope>/]<pkg> ...]
```

Description

This command will check the registry to see if any (or, specific) installed packages are currently outdated.

By default, only the direct dependencies of the root project and direct dependencies of your configured *workspaces* are shown. Use `--all` to find all outdated meta-dependencies as well.

In the output:

- `wanted` is the maximum version of the package that satisfies the semver range specified in `package.json`. If there's no available semver range (i.e. you're running `npm outdated --global`, or the package isn't included in `package.json`), then `wanted` shows the currently-installed version.
- `latest` is the version of the package tagged as latest in the registry. Running `npm publish` with no special configuration will publish the package with a dist-tag of `latest`. This may or may not be the maximum version of the package, or the most-recently published version of the package, depending on how the package's developer manages the latest dist-tag.
- `location` is where in the physical tree the package is located.
- `depended by` shows which package depends on the displayed dependency
- `package type` (when using `--long` / `-l`) tells you whether this package is a `dependency` or a dev/peer/optional dependency. Packages not included in `package.json` are always marked `dependencies`.
- `homepage` (when using `--long` / `-l`) is the `homepage` value contained in the package's packument
- Red means there's a newer version matching your semver requirements, so you should update now.
- Yellow indicates that there's a newer version *above* your semver requirements (usually new major, or new 0.x minor) so proceed with caution.

An example

```
$ npm outdated
Package      Current   Wanted   Latest  Location                  Depended by
glob          5.0.15   5.0.15    6.0.1  node_modules/glob         dependent-package-
name
nothingness    0.0.3      git      git  node_modules/nothingness  dependent-package-
name
npm            3.5.1    3.5.2    3.5.1  node_modules/npm          dependent-package-
name
local-dev      0.0.3   linked   linked  local-dev                 dependent-package-
name
once           1.3.2    1.3.3    1.3.3  node_modules/once         dependent-package-
name
```

With these `dependencies`:

```
{
  "glob": "^5.0.15",
  "nothingness": "github:othiym23/nothingness#master",
  "npm": "^3.5.1",
  "once": "^1.3.1"
}
```

A few things to note:

- `glob` requires `^5`, which prevents npm from installing `glob@6`, which is outside the semver range.
- Git dependencies will always be reinstalled, because of how they're specified. The installed committish might satisfy the dependency specifier (if it's something immutable, like a commit SHA), or it might not, so `npm outdated` and `npm update` have to fetch Git repos to check. This is why currently doing a reinstall of a Git dependency always forces a new clone and install.
- `npm@3.5.2` is marked as "wanted", but "latest" is `npm@3.5.1` because npm uses dist-tags to manage its `latest` and `next` release channels. `npm update` will install the *newest* version, but `npm install npm` (with no semver range) will install whatever's tagged as `latest`.
- `once` is just plain out of date. Reinstalling `node_modules` from scratch or running `npm update` will bring it up to spec.

Configuration

`all`

- Default: false
- Type: Boolean

When running `npm outdated` and `npm ls`, setting `--all` will show all outdated or installed packages, rather than only those directly depended upon by the current project.

`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.

`long`

- Default: false
- Type: Boolean

Show extended information in `ls`, `search`, and `help-search`.

`parseable`

- Default: false
- Type: Boolean

Output parseable results from commands that write to standard output. For `npm search`, this will be tab-separated table format.

`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 folders for more on the differences in behavior.

- 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`

`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.

See Also

- npm update
- npm dist-tag
- npm registry
- npm folders
- npm workspaces