



# API

## [Home](#)

## [Creating Packages](#)

## [API](#)

### [Commands](#)

### [Options](#)

### [Consuming a package](#)

### [Programmatic API](#)

### [Running on a continuous integration server](#)

### [Non-interactive mode](#)

### [Using local cache](#)

## [Configuration](#)

## [Pluggable Resolvers](#)

## [Tools](#)

## [About](#)

 [Bower on GitHub](#)

 [@bower](#)

 [Get support on discord](#)

 [Support on Bountysource](#)

## Commands

### Command line reference

- [cache](#)
- [help](#)
- [home](#)
- [info](#)
- [init](#)
- [install](#)
- [link](#)
- [list](#)
- [login](#)
- [lookup](#)
- [prune](#)
- [register](#)
- [search](#)
- [update](#)
- [uninstall](#)
- [unregister](#)
- [version](#)

## cache

```
$ bower cache <command> [<args>]
```

### Manage bower cache

#### cache clean

```
$ bower cache clean
$ bower cache clean <name> [<name>
...]
$ bower cache clean <name>#<version>
[<name>#<version> ..]
```

Cleans cached packages

## cache list

```
$ bower cache list
$ bower cache list <name> [<name> ...]
```

Lists cached packages

## help

```
$ bower help <command>
```

Display help information about Bower

## home

```
$ bower home
$ bower home <package>
$ bower home <package>#<version>
```

Opens a package homepage into your favorite browser.

If no <package> is passed, opens the homepage of the local package.

## info

```
$ bower info <package>
$ bower info <package> [<property>]
$ bower info <package>#<version>
[<property>]
```

Displays overall information of a package or of a particular version.

## init

```
$ bower init
```

Interactively create a bower.json file

## install

```
$ bower install [<options>]
$ bower install <endpoint> [<endpoint>
..] [<options>]
```

Installs project dependencies recursively.

Project dependencies consist of:

1. dependencies specified in bower.json of project
2. All “external” dependencies not specified in bower.json, but present in bower\_components
3. Any additional <endpoint> passed as an argument to this command

When --save flag is used, all additional endpoint are saved to dependencies in bower.json.

Bower recommends to always use --save flag to achieve reproducible installs between machines.

Endpoints can have multiple forms:

- `<package>`
- `<package>#<version>`
- `<name>=<package>#<version>`

Where:

- `<package>` is a package URL, physical location or registry name
- `<version>` is a valid range, commit, branch, etc.
- `<name>` is the name it should have locally.

`<package>` can be any one of the following:

Registered package name	jquery normalize.css
Git endpoint	https://github.com/user/package.git git@github.com:user/package.git
Git endpoint without .git	git+https://github.com/user/package git+ssh://git@github.com/user/package
Local folder	my/local/folder/
Public Subversion endpoint	svn+http://package.googlecode.com/svn/
Private Subversion endpoint	svn+ssh://package.googlecode.com/svn/ svn+https://package.googlecode.com/svn/
Shorthand (defaults to GitHub)	user/package
URL	http://example.com/script.js http://example.com/style.css http://example.com/package.zip (contents will be extracted) http://example.com/package.tar (contents will be extracted)

A version can be:

semver version	#1.2.3
version range	#1.2

	#~1.2.3
	^1.2.3
	>=1.2.3 <2.0
Git tag	#<tag>
Git commit SHA	#<sha>
Git branch	#<branch>
Subversion revision	#<revision>

## install options

- -F, --force-latest: Force latest version on conflict
- -p, --production: Do not install project devDependencies
- -S, --save: Save installed packages into the project's bower.json dependencies
- -D, --save-dev: Save installed packages into the project's bower.json devDependencies
- -E, --save-exact: Configure installed packages with an exact version rather than semver

## link

```
$ bower link
$ bower link <name> [<local name>]
```

The link functionality allows developers to easily test their packages. Linking is a two-step process.

Using 'bower link' in a project folder will create a global link. Then, in some other package, `bower link <name>` will create a link in the components folder pointing to the previously created link.

This allows you to easily test a package because changes will be reflected immediately. When the link is no longer necessary, simply remove it with `bower uninstall <name>`.

## list

```
$ bower list [<options>]
```

List local packages and possible updates.

### list options

- `-p, --paths`: Generates a simple JSON source mapping
- `-r, --relative`: Make paths relative to the directory config property, which defaults to `bower_components`

## lookup

```
$ bower lookup <name>
```

Look up a package URL by name

## login

```
$ bower login
```

Authenticate with GitHub and store credentials. Required to unregister packages.

### login options

- `-t, --token`: Pass an existing GitHub auth token rather than prompting for username and password

## prune

```
$ bower prune
```

Uninstalls local extraneous packages

## register

```
$ bower register <name> <url>
```

Register a package

## search

```
$ bower search  
$ bower search <name>
```

Finds all packages or a specific package.

## update

```
$ bower update <name> [<name> ..]  
[<options>]
```

Updates installed packages to their newest version according to bower.json.

### update options

- -F, --force-latest: Force latest version on conflict
- -p, --production: Do not install project devDependencies
- -S, --save: Update dependencies in bower.json
- -D, --save-dev: Update devDependencies in bower.json

## uninstall

```
$ bower uninstall <name> [<name> ..]  
[<options>]
```

Uninstalls a package locally from your `bower_components` directory

## uninstall options

- `-S, --save` : Remove uninstalled packages from the project's `bower.json` dependencies
- `-D, --save-dev` : Remove uninstalled packages from the project's `bower.json` `devDependencies`

## unregister

```
$ bower unregister <package>
```

Unregisters a package.

## version

```
$ bower version [<newversion> | major  
| minor | patch]
```

Run this in a package directory to bump the version and write the new data back to the `bower.json` file.

The `newversion` argument should be a valid semver string, or a valid second argument to `semver.inc` (one of “build”, “patch”, “minor”, or “major”). In the second case, the existing version will be incremented by 1 in the specified field.

If run in a git repo, it will also create a version commit and tag, and fail if the repo is not clean.

## version options

- `-m, --message` : Custom git commit and tag message

If supplied with `--message` (shorthand: `-m`) config option, bower 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:

```
$ bower version patch -m "Upgrade to  
%s for reasons"
```

## Options

- `force`
- `json`
- `loglevel`
- `offline`
- `quiet`
- `silent`
- `verbose`
- `allow-root`

## force

`-f, --force`

Makes various commands more forceful

- `bower install --force` re-installs all installed components. It also forces installation even when there are non-bower directories with the same name in the components directory. Adding `--force` also bypasses the cache, and writes to the cache anyway.
- `bower uninstall <package> --force` continues uninstallation even after a dependency conflict
- `bower register <package> --force` and `bower unregister <package> --force` bypasses confirmation. Login is still needed.

## json

`-j, --json`

Output consumable JSON

## loglevel

`-l, --loglevel`

What level of logs to report. Possible values: error, conflict, warn, action, info, debug

## offline

`-o, --offline`

Do not use network connection

## quiet

`-q, --quiet`

Only output important information. It is an alias for `--loglevel=warn`.

## silent

`-s, --silent`

Do not output anything, besides errors. It is an alias for `--loglevel=error`. Silent is also useful if you have private components that might leak credentials to your CI environment.

## verbose

```
-V, --verbose
```

Makes output more verbose. It is an alias for `--loglevel=debug`.

## allow-root

```
--allow-root
```

Allows running commands as root. Bower is a user command, there is no need to execute it with superuser permissions. However, if you still want to run commands with sudo, use `--allow-root` option.

# Consuming a package

You can use [build tools](#) to easily consume Bower packages.

If you use `bower list --paths` or `bower list --paths --json`, you will get a simple name-to-path mapping:

```
$ bower list --paths
# or
$ bower list --paths --json

{
  "backbone":
    "bower_components/backbone/backbone.js",
```

```

    "jquery":
    "bower_components/jquery/dist/jquery.js",

    "underscore":
    "bower_components/underscore/underscore.js"

  }

```

Every command supports the `--json` option that makes Bower output JSON. Command result is outputted to `stdout` and error/logs to `stderr`.

## Programmatic API

Bower provides a powerful, programmatic API. All commands can be accessed through the `bower.commands` object.

```

var bower = require('bower');

bower.commands
  .install(['jquery'], { save: true }, {
    /* custom config */ })
  .on('end', function (installed) {
    console.log(installed);
  });

bower.commands
  .search('jquery', {})
  .on('end', function (results) {
    console.log(results);
  });

```

Commands emit four types of events: `log`, `prompt`, `end`, `error`.

- `log` is emitted to report the state/progress of the command.

- `prompt` is emitted whenever the user needs to be prompted.
- `error` will only be emitted if something goes wrong.
- `end` is emitted when the command successfully ends.

For a better idea of how this works, you may want to check out [our bin file](#).

When using Bower programmatically, prompting is disabled by default. You can enable it when calling commands with `interactive: true` in the config. This requires you to listen for the `prompt` event and handle the prompting yourself. The easiest way is to use the [inquirer](#) npm module like so:

```
var inquirer = require('inquirer');

bower.commands
  .install(['jquery'], { save: true }, {
    interactive: true })
  // ..
  .on('prompt', function (prompts,
    callback) {
    inquirer.prompt(prompts,
      callback);
  });
```

## Running on a continuous integration server

Bower will skip some interactive operations if it finds a `CI` environmental variable set to `true`. You will find that the `CI` variable is already set for you on many continuous integration servers, e.g., [CircleCI](#) and [Travis-CI](#).

You may try to set the `CI` variable manually before running your Bower commands. On Mac or Linux, `export CI=true` and on Windows set `CI=true`

If for some reason you are unable to set the `CI` environment variable, you can alternately use the `--config.interactive=false` flag.

```
$ bower install --  
config.interactive=false
```

## Non-interactive mode

Bower works by default in interactive mode. There are few ways of disabling it:

- passing `CI=true` in environment
- passing `--config.interactive=false` to Bower command
- attaching a pipe to Bower (e.g. `bower install | cat`)
- redirecting output to file (e.g. `bower install > logs.txt`)
- running Bower through its [Programmatic API](#)

When interactive mode is disabled:

- `bower init` does not work
- `bower register` and `bower unregister` bypass confirmation
- `bower login` fails unless `--token` parameter is provided
- `bower install` fails on resolution conflicts, instead of asking for choice
- `bower uninstall` doesn't ask for confirmation if dependency is to be removed

## Using local cache

Bower supports installing packages from its local cache – without an internet connection – if the packages were installed before.

```
$ bower install <package> --offline
```

The content of the cache can be listed with `bower cache list`:

```
$ bower cache list
```

The cache can be cleaned with `bower cache clean`:

```
$ bower cache clean
```

---

[Help improve these docs. Open an issue or pull request.](#)