This package has been deprecated

Author message:

> angular-cli has been renamed to @angular/cli. Please update your dependencies.

angular-cli

1.0.0-beta.28.3 • Public • Published 2 years ago

Readme

67 Dependencies

8 Dependents

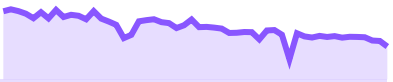
88 Versions

install

```
> npm i angular-cli
```

weekly downloads

7,131



version
1.0.0-beta.28.3

license
MIT

open issues
1420

pull requests
53

homepage
github.com

repository
github

last publish
2 years ago

collaborators

Test with RunKit | Report a vulnerability

Angular-CLI

Prototype of a CLI for Angular applications based on the ember-cli project.

Note

This project is very much still a work in progress.

The CLI is now in beta. If you wish to collaborate while the project is still young, check out our issue list.

Before submitting new issues, have a look at issues marked with the `type: faq` label.

Webpack update

We changed the build system between beta.10 and beta.14, from SystemJS to Webpack. And with it comes a lot of benefits. To take advantage of these, your app built with the old beta will need to migrate.

You can update your `beta.10` projects to `beta.14` by following these instructions.

Prerequisites

Both the CLI and generated project have dependencies that require Node 4 or higher, together with NPM 3 or higher.

Table of Contents

- Global styles
- CSS preprocessor integration
- 3rd Party Library Installation
- Global Library Installation
- Updating angular-cli
- Development Hints for hacking on angular-cli

Installation

BEFORE YOU INSTALL: please read the prerequisites

```
npm install -g angular-cli
```

Usage

```
ng help
```

Generating and serving an Angular2 project via a development server

```
ng new PROJECT_NAME
cd PROJECT_NAME
ng serve
```

Navigate to `http://localhost:4200/` . The app will automatically reload if you change any of the source files.

You can configure the default HTTP port and the one used by the LiveReload server with two command-line options :

```
ng serve --host 0.0.0.0 --port 4201 --live-reload-port 49153
```

Generating Components, Directives, Pipes and Services

You can use the `ng generate` (or just `ng g`) command to generate Angular components:

```
ng generate component my-new-component
ng g component my-new-component # using the alias

# components support relative path generation
# if in the directory src/app/feature/ and you run
ng g component new-cmp
# your component will be generated in src/app/feature/new-cmp
# but if you were to run
ng g component ../newer-cmp
# your component will be generated in src/app/newer-cmp
```

You can find all possible blueprints in the table below:

Scaffold	Usage
Component	`ng g component my-new-component`
Directive	`ng g directive my-new-directive`
Pipe	`ng g pipe my-new-pipe`
Service	`ng g service my-new-service`
Class	`ng g class my-new-class`
Interface	`ng g interface my-new-interface`
Enum	`ng g enum my-new-enum`
Module	`ng g module my-module`

Generating a route

The CLI supports routing in several ways:

- We include the `@angular/router` NPM package when creating or initializing a project.

- When you generate a module, you can use the `--routing` option like `ng g module my-module --routing` to create a separate file `my-module-routing.module.ts` to store the module routes.

  The file includes an empty `Routes` object that you can fill with routes to different components and/or modules.

  The `--routing` option also generates a default component with the same name as the module.

- You can use the `--routing` option with `ng new` to create a `app-routing.module.ts` file when you create or initialize a project.

Creating a build

```
ng build
```

The build artifacts will be stored in the `dist/` directory.

Build Targets and Environment Files

`ng build` can specify both a build target (`--target=production` or `--target=development`) and an environment file to be used with that build (`--environment=dev` or `--environment=prod`). By default, the development build target and environment are used.

The mapping used to determine which environment file is used can be found in `angular-cli.json` :

```
"environments": {
  "source": "environments/environment.ts",
  "dev": "environments/environment.ts",
  "prod": "environments/environment.prod.ts"
}
```

These options also apply to the serve command. If you do not pass a value for `environment`, it will default to `dev` for `development` and `prod` for `production`.

```
# these are equivalent
ng build --target=production --environment=prod
ng build --prod --env=prod
ng build --prod
# and so are these
ng build --target=development --environment=dev
ng build --dev --e=dev
ng build --dev
ng build
```

You can also add your own env files other than `dev` and `prod` by doing the following:

- create a `src/environments/environment.NAME.ts`
- add `{ "NAME": 'src/environments/environment.NAME.ts' }` to the `apps[0].environments` object in `angular-cli.json`
- use them via the `--env=NAME` flag on the build/serve commands.

Base tag handling in index.html

When building you can modify base tag (`<base href="/">`) in your index.html with `--base-href your-url` option.

```
# Sets base tag href to /myUrl/ in your index.html
ng build --base-href /myUrl/
ng build --bh /myUrl/
```

Bundling

All builds make use of bundling, and using the `--prod` flag in `ng build --prod` or `ng serve --prod` will also make use of uglifying and tree-shaking functionality.

Running unit tests

```
ng test
```

Tests will execute after a build is executed via Karma, and it will automatically watch your files for changes. You can run tests a single time via `--watch=false` or `--single-run`.

You can run tests with coverage via `--code-coverage`. The coverage report will be in the `coverage/` directory.

Linting during tests is also available via the `--lint` flag. See Linting code chapter for more information.

Running end-to-end tests

```
ng e2e
```

Before running the tests make sure you are serving the app via `ng serve`.

End-to-end tests are run via Protractor.

Proxy To Backend

Using the proxying support in webpack's dev server we can highjack certain urls and send them to a backend server. We do this by passing a file to `--proxy-config`

Say we have a server running on `http://localhost:3000/api` and we want all calls to `http://localhost:4200/api` to go to that server.

We create a file next to projects `package.json` called `proxy.conf.json` with the content

```
{
  "/api": {
    "target": "http://localhost:3000",
    "secure": false
  }
}
```

You can read more about what options are available here webpack-dev-server proxy settings

and then we edit the `package.json` file's start script to be

```
"start": "ng serve --proxy-config proxy.conf.json",
```

now run it with `npm start`

Deploying the app via GitHub Pages

You can deploy your apps quickly via:

```
ng github-pages:deploy --message "Optional commit message"
```

This will do the following:

- creates GitHub repo for the current project if one doesn't exist
- rebuilds the app in production mode at the current `HEAD`
- creates a local `gh-pages` branch if one doesn't exist
- moves your app to the `gh-pages` branch and creates a commit
- edit the base tag in index.html to support GitHub Pages
- pushes the `gh-pages` branch to GitHub
- returns back to the original `HEAD`

Creating the repo requires a token from GitHub, and the remaining functionality relies on ssh authentication for all git operations that communicate with github.com. To simplify the authentication, be sure to setup your ssh keys.

If you are deploying a user or organization page, you can instead use the following command:

```
ng github-pages:deploy --user-page --message "Optional commit message'
```

This command pushes the app to the `master` branch on the GitHub repo instead of pushing to `gh-pages` , since user and organization pages require this.

Linting code

You can lint your app code by running `ng lint`. This will use the `lint` npm script that in generated projects uses `tslint`.

You can modify the these scripts in `package.json` to run whatever tool you prefer.

Commands autocompletion

To turn on auto completion use the following commands:

For bash:

```
ng completion 1>> ~/.bashrc 2>>&1
source ~/.bashrc
```

For zsh:

```
ng completion 1>> ~/.zshrc 2>>&1
source ~/.zshrc
```

Windows users using gitbash:

```
ng completion 1>> ~/.bash_profile 2>>&1
source ~/.bash_profile
```

Project assets

You use the `assets` array in `angular-cli.json` to list files or folders you want to copy as-is when building your project:

```
"assets": [
  "assets",
  "favicon.ico"
]
```

Global styles

The `styles.css` file allows users to add global styles and supports CSS imports.

If the project is created with the `--style=sass` option, this will be a `.sass` file instead, and the same applies to `scss/less/styl`.

You can add more global styles via the `apps[0].styles` property in `angular-cli.json`.

CSS Preprocessor integration

Angular-CLI supports all major CSS preprocessors:

- sass/scss (http://sass-lang.com/)
- less (http://lesscss.org/)
- stylus (http://stylus-lang.com/)

To use these preprocessors simply add the file to your component's `styleUrls`:

```
@Component({
  selector: 'app-root',
  templateUrl: './app.component.html',
  styleUrls: ['./app.component.scss']
})
export class AppComponent {
  title = 'app works!';
}
```

When generating a new project you can also define which extension you want for style files:

```
ng new sassy-project --style=sass
```

Or set the default style on an existing project:

```
ng set defaults.styleExt scss
```

3rd Party Library Installation

Simply install your library via `npm install lib-name --save` and import it in your code.

If the library does not include typings, you can install them using npm:

```
npm install d3 --save
npm install @types/d3 --save-dev
```

If the library doesn't have typings available at `@types/`, you can still use it by manually adding typings for it:

1. First, create a `typings.d.ts` file in your `src/` folder. This file will be automatically included as global type definition.
2. Then, in `src/typings.d.ts`, add the following code:

```
declare module 'typeless-package';
```

1. Finally, in the component or file that uses the library, add the following code:

```
import * as typelessPackage from 'typeless-package';
typelessPackage.method();
```

Done. Note: you might need or find useful to define more typings for the library that you're trying to use.

Global Library Installation

Some javascript libraries need to be added to the global scope, and loaded as if they were in a script tag. We can do this using the `apps[0].scripts` and `apps[0].styles` properties of `angular-cli.json`.

As an example, to use Bootstrap 4 this is what you need to do:

First install Bootstrap from `npm`:

```
npm install bootstrap@next
```

Then add the needed script files to `apps[0].scripts`:

```
"scripts": [
  "../node_modules/jquery/dist/jquery.js",
  "../node_modules/tether/dist/js/tether.js",
  "../node_modules/bootstrap/dist/js/bootstrap.js"
],
```

Finally add the Bootstrap CSS to the `apps[0].styles` array:

```
"styles": [
  "../node_modules/bootstrap/dist/css/bootstrap.css",
  "styles.css"
],
```

Restart `ng serve` if you're running it, and Bootstrap 4 should be working on your app.

Updating angular-cli

To update `angular-cli` to a new version, you must update both the global package and your project's local package.

Global package:

```
npm uninstall -g angular-cli
npm cache clean
npm install -g angular-cli@latest
```

Local project package:

```
rm -rf node_modules dist # use rmdir on Windows
npm install --save-dev angular-cli@latest
npm install
ng update
```

Running `ng update` will check for changes in all the auto-generated files created by `ng new` and allow you to update yours. You are offered four choices for each changed file: `y` (overwrite), `n` (don't overwrite), `d` (show diff between your file and the updated file) and `h` (help).

Carefully read the diffs for each code file, and either accept the changes or incorporate them manually after `ng update` finishes.

The main cause of errors after an update is failing to incorporate these updates into your code.

You can find more details about changes between versions in CHANGELOG.md.

Development Hints for hacking on angular-cli

Working with master

```
git clone https://github.com/angular/angular-cli.git
cd angular-cli
npm link
```

`npm link` is very similar to `npm install -g` except that instead of downloading the package from the repo, the just cloned `angular-cli/` folder becomes the global package. Any changes to the files in the `angular-cli/` folder will immediately affect the global `angular-cli` package, allowing you to quickly test any changes you make to the cli project.

Now you can use `angular-cli` via the command line:

```
ng new foo
cd foo
npm link angular-cli
ng serve
```

`npm link angular-cli` is needed because by default the globally installed `angular-cli` just loads the local `angular-cli` from the project which was fetched remotely from npm. `npm link angular-cli` symlinks the global `angular-cli` package to the local `angular-cli` package. Now the `angular-cli` you cloned before is in three places: The folder you cloned it into, npm's folder where it stores global packages and the `angular-cli` project you just created.

You can also use `ng new foo --link-cli` to automatically link the `angular-cli` package.

Please read the official npm-link documentation and the npm-link cheatsheet for more information.

License

MIT

Keywords

ember-addon

You Need Help

Documentation
Support / Contact Us
Registry Status
Report Issues

npm Community Site
Security

About npm

About npm, Inc
Jobs
npm Weekly
Blog
Twitter
GitHub

Terms & Policies

Terms of Use
Code of Conduct
Package Name Disputes
Privacy Policy
Reporting Abuse
Other policies

npm loves you