Angular cli commands:

Ng new <project>:

To create new project

## ng generate:

## generate a new component. You can also use ng generate directive|pipe|service|class|guard|interface|enum|module

**ex:**

ng g m timecard/timesmodule –routing

ng g component foo/fooList --module=foo

ng build:

It compiles an Angular app into an output directory named dist/ at the given output path.

## Ng build –prod //build in production enviranment

ng serve:

for build and run the application.

## ng test:

## to execute the unit tests via [Karma](https://karma-runner.github.io/).

## ng e2e:

## It is used to build and serve an Angular app, then runs end-to-end tests using Protractor.

## Ng lint

## ng lint run the linting tool on angular app code. It checks the code quality of angular project specified. It uses TSLint as default linting tool and uses the default configuration available in tslint.json file

## Ng – version

## Get version of angular application

node -v  
npm –v

ng --version

## npm init:

## for create package.json file

## npm install:

## it install packages from package.json in  node\_modules is the folder

There will be a new file named package-lock.json. This file contains the exact version of the package, unlike package.json which contains the semantic version

You can also install packages as a developer dependency i.e., these packages are only needed for development

npm install <package\_name> --save-dev

A globally installed packages works anywhere on the machine. To install global packages you’ve to use -g flag.

Since we have installed packages sometimes we need to update our packages to get new features.

npm update <package\_name> for specifc package

npm update //for all packages

uninstall package:

npm uninstall <package\_name> -g

npm uninstall <package\_name>

To get the list of installed packages, use the command

npm list

to install specific package :

npm install <package\_name>@version

help:

npm help

NPM commands:

Flow:

main.ts  **>>**   app.Module.ts  **>>**  app.component.ts  **>>**  index.html  **>>**  app.component.html

**Step 1:** main.ts file gets loaded. It bootstraps(starts) the application by calling App.module.file.

**Step 2:** app.module.ts file holds an array of bootstrap components. Here, we find our root component reference.

**Step 3:** Root component gets loaded and the template files from app.component.html become part of index.html.

|  |  |
| --- | --- |
| ngOnChanges() | This event called every time when a value of an input control within the component has been changed. The method receives a [SimpleChanges](https://angular.io/api/core/SimpleChanges) object of current and previous property values.  Called before ngOnInit() and whenever one or more data-bound input properties change. |
| ngOnInit() | Initialize the directive/component after Angular first displays the data-bound properties and sets the directive/component's input properties.  Called *once*, after the *first*ngOnChanges(). |
| ngDoCheck() | Detect changes when Angular can't detect on its own.  Called during every change detection run, immediately after ngOnChanges()and ngOnInit(). |
| [ngAfterContentInit()](https://angular.io/api/router/RouterLinkActive#ngAfterContentInit) | Called after Angular projects external content into the component's view / the view that a directive is in.  Called *once* after the first ngDoCheck(). |
| ngAfterContentChecked() | Called after Angular checks the content projected into the directive/component.  Called after the [ngAfterContentInit()](https://angular.io/api/router/RouterLinkActive" \l "ngAfterContentInit) and every subsequent ngDoCheck(). |
| [ngAfterViewInit()](https://angular.io/api/forms/NgForm#ngAfterViewInit) | Called after Angular initializes the component's views and child views / the view that a directive is in.  Called *once* after the first ngAfterContentChecked(). |
| ngAfterViewChecked() | Called after Angular checks the component's views and child views / the view that a directive is in.  Called after the [ngAfterViewInit()](https://angular.io/api/forms/NgForm" \l "ngAfterViewInit)and every subsequent ngAfterContentChecked(). |
| ngOnDestroy() | Cleanup just before Angular destroys the directive/component. Unsubscribe Observables and detach event handlers to avoid memory leaks.  Called *just before* Angular destroys the directive/component. |

BrowserModule @angular/platform-browser When you want to run your app in a browser

componentModule

NgModule

Injectable

Directive

Pipe

CommonModule @angular/common When you want to use NgIf, NgFor

FormsModule @angular/forms When you want to build template driven forms (includes NgModel)

ReactiveFormsModule @angular/forms When you want to build reactive forms

RouterModule, Routes @angular/router When you want to use RouterLink, .forRoot(), and .forChild()

HttpClientModule @angular/common/http When you want to talk to a server

needed for reactive forms : import{FormGroup,FormControl,Validators,FormBuilder} from from '@angular/forms';

Routing:

Routing helps in directing users to different pages based on the option they choose on the main page. Hence,

based on the option they choose, the required Angular Component will be rendered to the user.

<router-outlet> </router-outlet>.--container for display route views, it is directive from routermodule

forRoot creates a module that contains all the directives, the given routes, and the router service itself.

forChild creates a module that contains all the directives and the given routes, but does not include the

router service. It registers the routers and uses the router service created at the root level.

Angular 2:

Angular 2 is components based

used for develop mobile driven apps

no longer controllers and $scope

Support for TypeScript

multimple browser support

Angular 4:

Faster and smalle

template is now ng-template

httpclient

useses routermodule- forroute, forchild for routing

Angular 5:

Router Life Cycle

compiler improvements

RxJs Support

Angular 6:

Angular Elements

RxJs 6 Support

Subjects are specific kind of observable and observer.

A subject is observable,so we can get values from the observable stream via subscribe i.e We can subscribe to them.

they support multiple subscriptions. In other words, they are multicast.but observables are singlecast.

A subject is observer,so we can update the observable stream with next(),error, complete method

but regular observable does not.

Behavior Subjects: When we subscribe to a behavior subject, it will give you the last emitted value right away.

Replay Subjects: Replay Subjects keep a given number of historical values so that those values can be replayed to new subscribers.

https://coursetro.com/posts/code/149/RxJS-Subjects-Tutorial---Subjects,-BehaviorSubject,-ReplaySubject-&-AsyncSubject

https://medium.com/@luukgruijs/understanding-rxjs-behaviorsubject-replaysubject-and-asyncsubject-8cc061f1cfc0