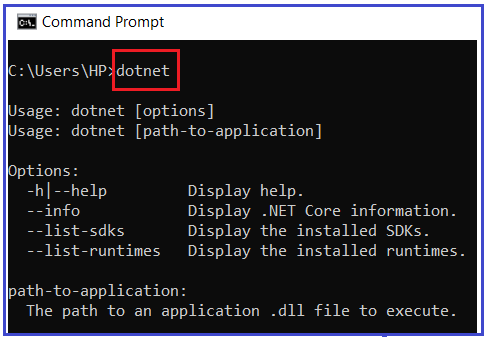
**ASP.NET Core Command Line Interface (.NET Core CLI)**

##### **ASP.NET Core Command Line Interface:**

The .NET Core CLI (Command Line Interface) is a new cross-platform tool that is used for creating, restoring packages, building, running, and publishing ASP.NET Core Applications. The .NET Core CLI command for any kind of web application uses Out of Process hosting i.e. it uses the Kestrel server to run the application.

As of now, all the applications we created are using Visual Studio. Visual Studio internally uses this .NET CLI commands to restore, build, and publish the applications. Other higher-level IDEs, editors, and tools, for example, Visual Studio Code uses these CLI Commands to support to create, restore, publish, and run .NET Core applications.

When we installed .NET Core SDK, then by default the .NET Core CLI is also installed. So, we don’t require installing it separately on the development environment i.e. on our local machine. We can verify the same i.e. whether the .NET CLI is installed or not using the command prompt. To verify the same, open command prompt (Windows), terminal (Linux) and type “**dotnet**” and press enter as shown in the below. If it displays usage and helps options as you can see in the below image means the .NET Core CLI is installed properly.



##### **.NET Core CLI Command Structure:**

The .NET Core CLI command structure is nothing but how we write the .NET Core CLI command. The following is the command structure of .NET Core CLI Command:

**dotnet <command> <argument> <option>**

**Note:** All the .NET Core CLI commands start with the driver named **dotnet**. The driver i.e. dotnet starts the execution of the specified command. After dotnet, we need to specify the **command** (also known as the verb) to perform a specific action. Each command can be followed by **arguments** and **options**.

##### **How to get the all the .NET Core Commands:**

Open the command prompt and type **dotnet help** and press enter which will display all the .NET Core CLI commands. Some of the command and their use is given below.

1. **add**: Add a package or reference to a .NET project.
2. **build**: Build a .NET project.
3. **build-server**: Interact with servers started by a build.
4. **clean**: Clean build outputs of a .NET project.
5. **help**: Show command-line help.
6. **list:** List project references for a .NET project.
7. **msbuild:** Run Microsoft Build Engine (MSBuild) commands.
8. **new**: Create a new .NET project or file.
9. **nuget**: Provides additional NuGet commands.
10. **pack**: Create a NuGet package.
11. **publish**: Publish a .NET project for deployment.
12. **remove**: Remove a package or reference from a .NET project.
13. **restore**: Restore dependencies specified in a .NET project.
14. **run**: Build and run a .NET project output.
15. **sln**: Modify Visual Studio solution files.
16. **store**: Store the specified assemblies in the runtime package store.
17. **test**: Run unit tests using the test runner specified in a .NET project.
18. **tool**: Install or manage tools that extend the .NET experience.
19. **vstest**: Run Microsoft Test Engine (VSTest) commands.

##### **Project Modification Commands:**

1. **add package:** Adds a package reference to a project.
2. **add reference:** Adds project-to-project (P2P) references.
3. **remove package:** Removes package reference from the project.
4. **remove reference:** Removes project reference.
5. **list reference:** Lists all project-to-project references.

##### **Advanced Command:**

1. **nuget delete:** Deletes or un-lists a package from the server.
2. **nuget locals:** Clear or lists NuGet resources.
3. **nuget push:** Pushes a package to the server and publishes it.
4. **msbuild:** Builds a project and all of its dependencies.
5. dotnet install script: Script used to install .NET Core CLI tools and the shared runtime.

##### **Create a New Project using the .NET Core CLI Command:**

Let’s create, restore, build, and run .NET Core console application using the command-line interface without using Visual Studio. To create a new .NET Core project, we have to use the ‘new’ command followed by the template name argument. We can create the console, class library, web, webapp, mvc, webapi, razor, angular, react, etc. project using CLI.

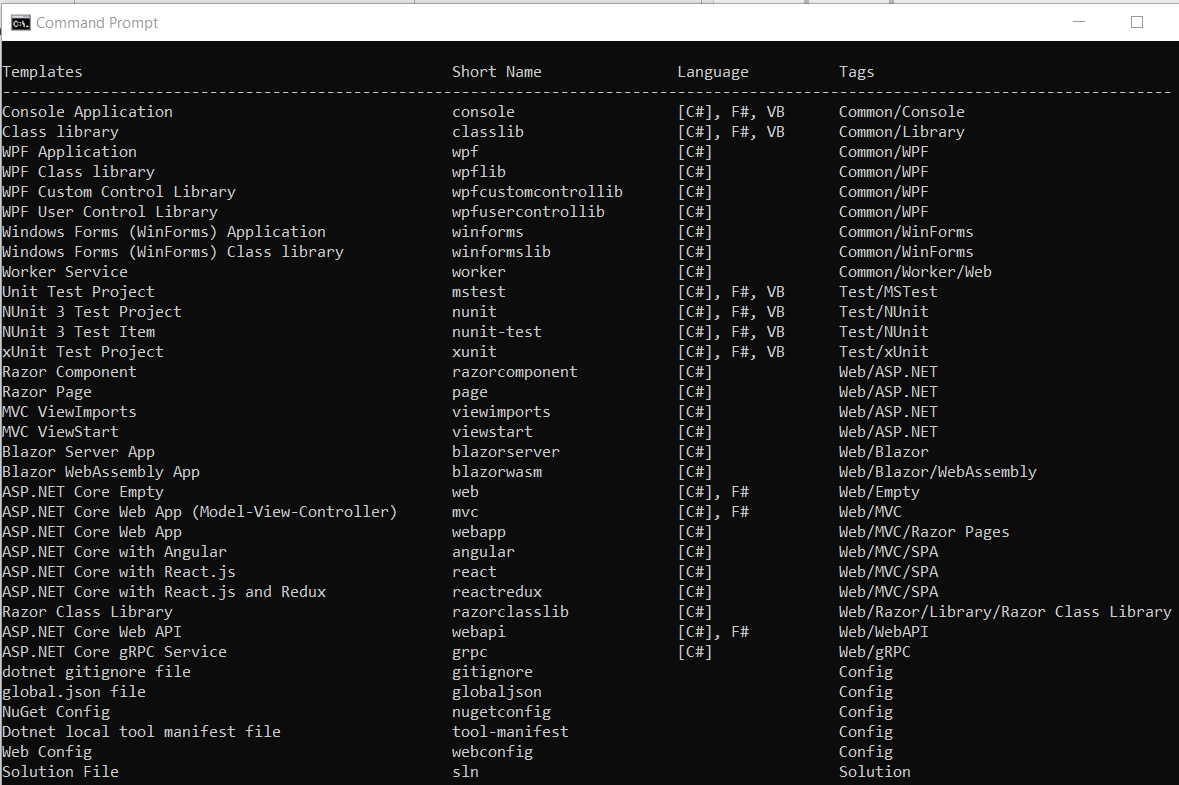
The following command creates a new dotnet core project using the TEMPLATE:

**dotnet new <TEMPLATE>**

You can find out the list of templates using:

**dotnet new -l**

Once you type **dotnet new -l**and press enter, it will show you the list of available templates based on .NET Core Version installed on your machine as shown in the below image:

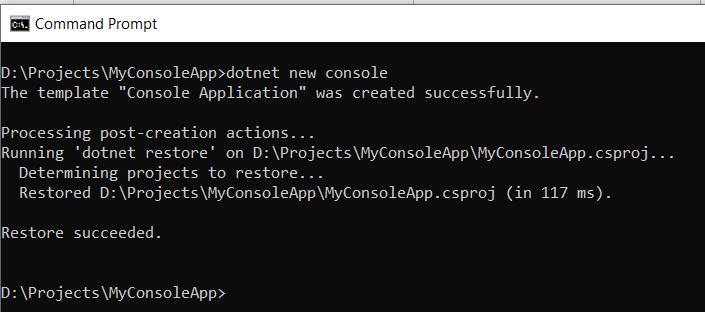


##### **Example: Creating a Console Application using .NET Core CLI**

The following command creates a new console project in the current directory with the same name as the current directory.

**D:\Projects\MyConsoleApp>dotnet new console**

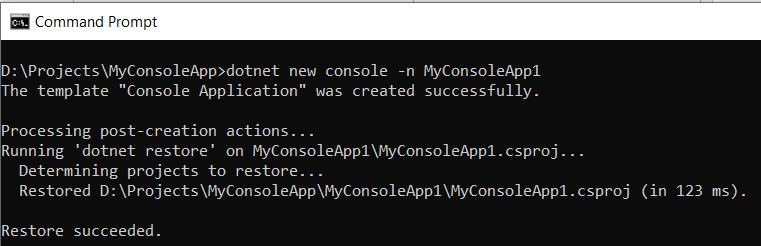
Once you execute the above command, it will create a console application and you will get the following output.



The following command creates a new console project named **‘MyConsoleApp1**“. The **-n or –name** option specifies the name of the project.

**D:\DotNetCoreApps>dotnet new console -n MyConsoleApp1**

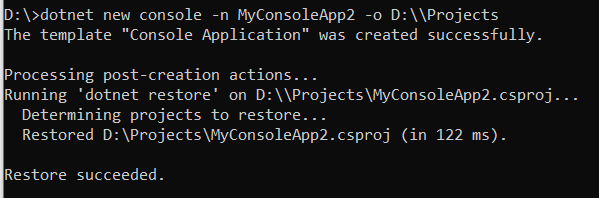
Once you execute the above command, it will create a new console application with the name MyConsoleApp1 and you will get the following output.



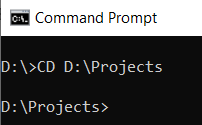
The following command creates a new console application named MyConsoleApp2 to \Projects directory. The **-o or –output** option is used to specify an output directory where the project should be generated.

**dotnet new console -n MyConsoleApp2 -o D:\\Projects**

Once you execute the above command, it will create a new console application with the name MyConsoleApp2 with the D=>Projects directory and you will get the following output.



After creating a project, navigate to the project directory (folder) in the command prompt to apply project-specific commands. As we created the project in **D=>Projects** folder, then go to the **D => Projects** directory in the command prompt as shown below.

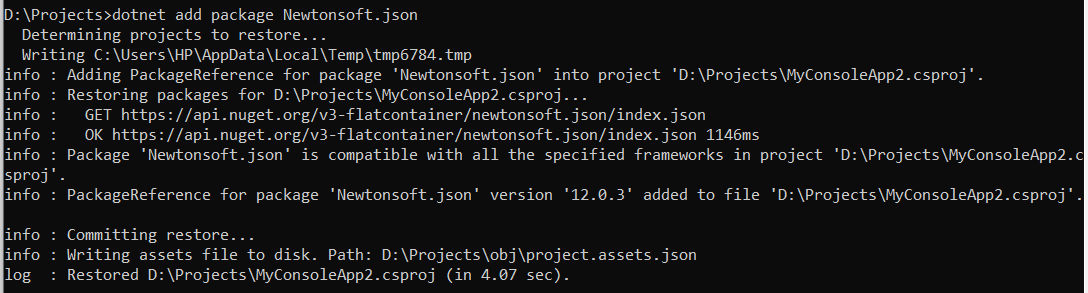


##### **Add Package Reference using .NET Core CLI Command:**

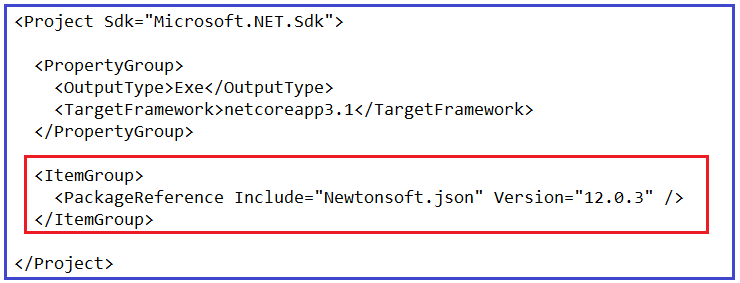
We often need to add the NuGet package references for different purposes. For example, apply the following command to add **Newtonsoft.json** package to your console project.

**dotnet add package Newtonsoft.json**

Once you type the above command and press enter then you should get the following output.



This will add the Newtonsoft.json package to your project. You can verify the same in the project file. So, open .csproj file and you should get the following.



##### **Remove Package Reference using .NET Core CLI Command:**

The “**dotnet remove package**” command provides a convenient option to remove a NuGet package reference from a project. If you want to remove the NuGet Package that we just installed Newtonsoft.json then you need to execute the below command,

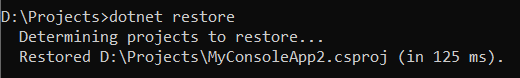
**dotnet remove package Newtonsoft.json**

So, in the command prompt and type “**dotnet remove package Newtonsoft.json**” and press enter as shown in the below image which will remove the Newtonsoft.json package from your project and you verify the same in the project file.

Remove Package Reference using .NET Core CLI Command

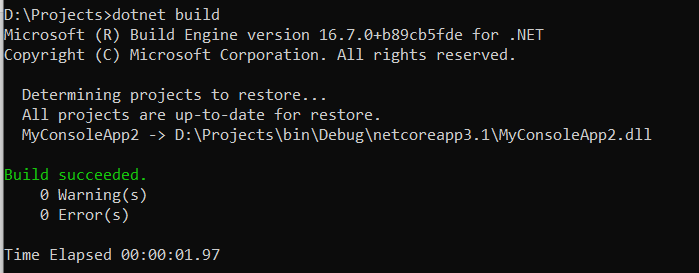
##### **Restore Packages using .NET Core CLI Command:**

To restore packages or to update existing packages in your project, you can use the “**dotnet restore**” command as below:



##### **Build Project using .NET Core CLI Command:**

In order to build a new or existing project, we need to use the “**dotnet build**” command as below which will build your .NET Core Project:



##### **Run .NET Core Project using .NET Core CLI Command:**

To run the .NET Core project, we need to use “**dotnet run**” command as shown below: Here, you can see it display the output Hello World!

Run .NET Core Project using .NET Core CLI Command