

dotnet CLI

Microsoft - dotnet CLI should be all the information you need, but it helps me to learn new things if I take notes on them. Often I end up exploring things I might not have otherwise, and having the notes around is a good reference for me if I need to do something similar in the future.

To create a new C# project we can run the following commands from a bash terminal

```
mkdir dotnet-scrap
cd dotnet-scrap
dotnet new sln
```

The project structure is now

```
tree

.
└─ dotnet-scrap.sln
```

The solution file contains the following information

```
Microsoft Visual Studio Solution File, Format Version 12.00
# Visual Studio Version 16
VisualStudioVersion = 16.0.30114.105
MinimumVisualStudioVersion = 10.0.40219.1
Global
    GlobalSection(SolutionConfigurationPlatforms) = preSolution
        Debug|Any CPU = Debug|Any CPU
        Release|Any CPU = Release|Any CPU
    EndGlobalSection
    GlobalSection(SolutionProperties) = preSolution
        HideSolutionNode = FALSE
    EndGlobalSection
EndGlobal
```

This project doesn't have any `.cs` files though, so there isn't anything to build or run at this time. In the next sections I will show how to create different projects and link them together for use.

Custom Library

Then we create a library, called `ScrapLibrary`

```
dotnet new classlib -o ScrapLibrary
```

The template "Class Library" was created successfully.

Processing post-creation actions...

Running 'dotnet restore' on /home/kapper/Code/dotnet-scrap/ScrapLibrary/ScrapLibrary.csproj...

Determining projects to restore...

Restored /home/kapper/Code/dotnet-scrap/ScrapLibrary/ScrapLibrary.csproj (in 63 ms).

Restore succeeded.

The project structure is now

```
tree -L 2
```

```
.
├─ dotnet-scrap.sln
└─ ScrapLibrary
   ├─ Class1.cs
   ├─ obj
   └─ ScrapLibrary.csproj
```

2 directories, 8 files

Now we add the library to our project solution in the root directory of `dotnet-scrap`

```
dotnet sln add ScrapLibrary/ScrapLibrary.csproj
```

The `dotnet-scrap.sln` file now contains the following information. The project file structure has not changed.

Microsoft Visual Studio Solution File, Format Version 12.00

Visual Studio Version 16

VisualStudioVersion = 16.0.30114.105

MinimumVisualStudioVersion = 10.0.40219.1

Project("{FAE04EC0-301F-11D3-BF4B-00C04F79EFBC}") = "ScrapLibrary", "ScrapLibrary\ScrapLibrary.csproj", "{225F8AD6-761E-436E-9BDE-DC5D3490C38E}"

EndProject

Global

```

GlobalSection(SolutionConfigurationPlatforms) = preSolution
    Debug|Any CPU = Debug|Any CPU
    Release|Any CPU = Release|Any CPU
EndGlobalSection
GlobalSection(SolutionProperties) = preSolution
    HideSolutionNode = FALSE
EndGlobalSection
GlobalSection(ProjectConfigurationPlatforms) = postSolution
    {225F8AD6-761E-436E-9BDE-DC5D3490C38E}.Debug|Any CPU.ActiveCfg = Debug|Any CPU
    {225F8AD6-761E-436E-9BDE-DC5D3490C38E}.Debug|Any CPU.Build.0 = Debug|Any CPU
    {225F8AD6-761E-436E-9BDE-DC5D3490C38E}.Release|Any CPU.ActiveCfg = Release|Any CPU
    {225F8AD6-761E-436E-9BDE-DC5D3490C38E}.Release|Any CPU.Build.0 = Release|Any CPU
EndGlobalSection
EndGlobal

```

And for later testing, we define a `public static` function that we can call to verify the library is available within later projects. This is done in the `ScrapLibrary/Class1.cs` file.

```

namespace ScrapLibrary;
public class Class1
{
    public static void Hello()
    {
        Console.WriteLine("Hello!\n");
    }
}

```

XUnit Unit Testing

Now if we want to add the `xunit` testing framework to our project, we run the following command

```
dotnet new xunit -o ScrapLibrary.Test
```

The file structure of our project is now

```

tree -L 2
.
├─ dotnet-scrap.sln
├─ ScrapLibrary
│   └─ Class1.cs
└─ obj

```

```
| └─ ScrapLibrary.csproj
└─ ScrapLibrary.Test
    └─ obj
        └─ ScrapLibrary.Test.csproj
            └─ UnitTest1.cs
```

4 directories, 5 files

But we still need to add the `ScrapLibrary.test` to our `dotnet-scrap.sln`, and to do so we run the following command

```
dotnet sln add ScrapLibrary.Test/ScrapLibrary.Test.csproj
```

This updates our `dotnet-scrap.sln` with the following information

```
Microsoft Visual Studio Solution File, Format Version 12.00
# Visual Studio Version 16
VisualStudioVersion = 16.0.30114.105
MinimumVisualStudioVersion = 10.0.40219.1
Project("{FAE04EC0-301F-11D3-BF4B-00C04F79EFBC}") = "ScrapLibrary", "ScrapLibrary\ScrapLibrary.csproj",
"{225F8AD6-761E-436E-9BDE-DC5D3490C38E}"
EndProject
Project("{FAE04EC0-301F-11D3-BF4B-00C04F79EFBC}") = "ScrapLibrary.Test",
"ScrapLibrary.Test\ScrapLibrary.Test.csproj", "{85594F96-1E54-4BF6-98B8-3B2C7861B308}"
EndProject
Global
    GlobalSection(SolutionConfigurationPlatforms) = preSolution
        Debug|Any CPU = Debug|Any CPU
        Release|Any CPU = Release|Any CPU
    EndGlobalSection
    GlobalSection(SolutionProperties) = preSolution
        HideSolutionNode = FALSE
    EndGlobalSection
    GlobalSection(ProjectConfigurationPlatforms) = postSolution
        {225F8AD6-761E-436E-9BDE-DC5D3490C38E}.Debug|Any CPU.ActiveCfg = Debug|Any CPU
        {225F8AD6-761E-436E-9BDE-DC5D3490C38E}.Debug|Any CPU.Build.0 = Debug|Any CPU
        {225F8AD6-761E-436E-9BDE-DC5D3490C38E}.Release|Any CPU.ActiveCfg = Release|Any CPU
        {225F8AD6-761E-436E-9BDE-DC5D3490C38E}.Release|Any CPU.Build.0 = Release|Any CPU
        {85594F96-1E54-4BF6-98B8-3B2C7861B308}.Debug|Any CPU.ActiveCfg = Debug|Any CPU
        {85594F96-1E54-4BF6-98B8-3B2C7861B308}.Debug|Any CPU.Build.0 = Debug|Any CPU
        {85594F96-1E54-4BF6-98B8-3B2C7861B308}.Release|Any CPU.ActiveCfg = Release|Any CPU
```

```
{85594F96-1E54-4BF6-98B8-3B2C7861B308}.Release|Any CPU.Build.0 = Release|Any CPU
EndGlobalSection
EndGlobal
```

Initially, the `ScrapProject.Test.csproj` will not contain a reference to our `ScrapLibrary` project.

```
cat ScrapLibrary.Test.csproj

<Project Sdk="Microsoft.NET.Sdk">

  <PropertyGroup>
    <TargetFramework>net6.0</TargetFramework>
    <Nullable>enable</Nullable>

    <IsPackable>>false</IsPackable>
  </PropertyGroup>

  <ItemGroup>
    <PackageReference Include="Microsoft.NET.Test.Sdk" Version="16.11.0" />
    <PackageReference Include="xunit" Version="2.4.1" />
    <PackageReference Include="xunit.runner.visualstudio" Version="2.4.3">
      <IncludeAssets>runtime; build; native; contentfiles; analyzers; buildtransitive</IncludeAssets>
      <PrivateAssets>all</PrivateAssets>
    </PackageReference>
    <PackageReference Include="coverlet.collector" Version="3.1.0">
      <IncludeAssets>runtime; build; native; contentfiles; analyzers; buildtransitive</IncludeAssets>
      <PrivateAssets>all</PrivateAssets>
    </PackageReference>
  </ItemGroup>

</Project>
```

So we need to add a reference from our `ScrapLibrary.Test` project to our `ScrapLibrary` project, which will allow the use of our custom library.

```
cd ScrapLibrary.Test
dotnet add reference ../ScrapLibrary/ScrapLibrary.csproj
```

This updates our `ScrapLibrary.Test.csproj` to contain the following information

```

<Project Sdk="Microsoft.NET.Sdk">

  <PropertyGroup>
    <TargetFramework>net6.0</TargetFramework>
    <Nullable>enable</Nullable>

    <IsPackable>>false</IsPackable>
  </PropertyGroup>

  <ItemGroup>
    <PackageReference Include="Microsoft.NET.Test.Sdk" Version="16.11.0" />
    <PackageReference Include="xunit" Version="2.4.1" />
    <PackageReference Include="xunit.runner.visualstudio" Version="2.4.3">
      <IncludeAssets>runtime; build; native; contentfiles; analyzers; buildtransitive</IncludeAssets>
      <PrivateAssets>all</PrivateAssets>
    </PackageReference>
    <PackageReference Include="coverlet.collector" Version="3.1.0">
      <IncludeAssets>runtime; build; native; contentfiles; analyzers; buildtransitive</IncludeAssets>
      <PrivateAssets>all</PrivateAssets>
    </PackageReference>
  </ItemGroup>

  <ItemGroup>
    <ProjectReference Include="..\ScrapLibrary\ScrapLibrary.csproj" />
  </ItemGroup>

</Project>

```

And within the xunit `ScrapLibrary.Test/UnitTest1.cs` file we can now use the library -

```

using Xunit;

namespace ScrapLibrary.Test;

public class UnitTest1
{
    [Fact]
    public void Test1()
    {
        ScrapLibrary.Class1.Hello();
    }
}

```

```
}  
}
```

Console Application

And finally, as a last step for getting started using C#, we create a simple console application that will build within our solution, and have the ability to use our custom library!

```
dotnet new console -o ScrapConsole
```

The template "Console App" was created successfully.

Processing post-creation actions...

Running 'dotnet restore' on /home/kapper/Code/dotnet-scrap/ScrapConsole/ScrapConsole.csproj...

Determining projects to restore...

Restored /home/kapper/Code/dotnet-scrap/ScrapConsole/ScrapConsole.csproj (in 73 ms).

Restore succeeded.

Our project structure is now

```
tree -L 2
```

```
.  
├── dotnet-scrap.sln  
├── ScrapConsole  
│   ├── obj  
│   ├── Program.cs  
│   └── ScrapConsole.csproj  
├── ScrapLibrary  
│   ├── Class1.cs  
│   ├── obj  
│   └── ScrapLibrary.csproj  
└── ScrapLibrary.Test  
    ├── obj  
    ├── ScrapLibrary.Test.csproj  
    └── UnitTest1.cs
```

6 directories, 7 files

And we again add this `ScrapConsole` to our `dotnet-scrap.sln` just as we did previously for the library and xunit test projects.

```
dotnet sln add ScrapConsole/ScrapConsole.csproj
```

```
Project `ScrapConsole/ScrapConsole.csproj` added to the solution.
```

Which updates our `dotnet-scrap.sln` to contain the following information

```
Microsoft Visual Studio Solution File, Format Version 12.00
# Visual Studio Version 16
VisualStudioVersion = 16.0.30114.105
MinimumVisualStudioVersion = 10.0.40219.1
Project("{FAE04EC0-301F-11D3-BF4B-00C04F79EFBC}") = "ScrapLibrary", "ScrapLibrary\ScrapLibrary.csproj",
"{225F8AD6-761E-436E-9BDE-DC5D3490C38E}"
EndProject
Project("{FAE04EC0-301F-11D3-BF4B-00C04F79EFBC}") = "ScrapLibrary.Test",
"ScrapLibrary.Test\ScrapLibrary.Test.csproj", "{85594F96-1E54-4BF6-98B8-3B2C7861B308}"
EndProject
Project("{FAE04EC0-301F-11D3-BF4B-00C04F79EFBC}") = "ScrapConsole", "ScrapConsole\ScrapConsole.csproj",
"{21EAD6BF-28D7-40E9-BD11-FA8393FCBCF7}"
EndProject
Global
    GlobalSection(SolutionConfigurationPlatforms) = preSolution
        Debug|Any CPU = Debug|Any CPU
        Release|Any CPU = Release|Any CPU
    EndGlobalSection
    GlobalSection(SolutionProperties) = preSolution
        HideSolutionNode = FALSE
    EndGlobalSection
    GlobalSection(ProjectConfigurationPlatforms) = postSolution
        {225F8AD6-761E-436E-9BDE-DC5D3490C38E}.Debug|Any CPU.ActiveCfg = Debug|Any CPU
        {225F8AD6-761E-436E-9BDE-DC5D3490C38E}.Debug|Any CPU.Build.0 = Debug|Any CPU
        {225F8AD6-761E-436E-9BDE-DC5D3490C38E}.Release|Any CPU.ActiveCfg = Release|Any CPU
        {225F8AD6-761E-436E-9BDE-DC5D3490C38E}.Release|Any CPU.Build.0 = Release|Any CPU
        {85594F96-1E54-4BF6-98B8-3B2C7861B308}.Debug|Any CPU.ActiveCfg = Debug|Any CPU
        {85594F96-1E54-4BF6-98B8-3B2C7861B308}.Debug|Any CPU.Build.0 = Debug|Any CPU
        {85594F96-1E54-4BF6-98B8-3B2C7861B308}.Release|Any CPU.ActiveCfg = Release|Any CPU
        {85594F96-1E54-4BF6-98B8-3B2C7861B308}.Release|Any CPU.Build.0 = Release|Any CPU
        {21EAD6BF-28D7-40E9-BD11-FA8393FCBCF7}.Debug|Any CPU.ActiveCfg = Debug|Any CPU
        {21EAD6BF-28D7-40E9-BD11-FA8393FCBCF7}.Debug|Any CPU.Build.0 = Debug|Any CPU
        {21EAD6BF-28D7-40E9-BD11-FA8393FCBCF7}.Release|Any CPU.ActiveCfg = Release|Any CPU
        {21EAD6BF-28D7-40E9-BD11-FA8393FCBCF7}.Release|Any CPU.Build.0 = Release|Any CPU
```



```
EndGlobalSection
EndGlobal
```

Initially, the contents of `ScrapConsole/ScrapConsole.csproj` will not contain a reference to our `ScrapLibrary`

```
cat ScrapConsole.csproj

<Project Sdk="Microsoft.NET.Sdk">

  <PropertyGroup>
    <OutputType>Exe</OutputType>
    <TargetFramework>net6.0</TargetFramework>
    <ImplicitUsings>enable</ImplicitUsings>
    <Nullable>enable</Nullable>
  </PropertyGroup>

</Project>
```

So we add a reference from our `ScrapConsole` project to allow use of our custom `ScrapLibrary`

```
cd ScrapConsole
dotnet add reference ../ScrapLibrary/ScrapLibrary.csproj
```

And now the `ScrapConsole.csproj` contains the following information

```
cat ScrapConsole.csproj

<Project Sdk="Microsoft.NET.Sdk">

  <ItemGroup>
    <ProjectReference Include="..\ScrapLibrary\ScrapLibrary.csproj" />
  </ItemGroup>

  <PropertyGroup>
    <OutputType>Exe</OutputType>
    <TargetFramework>net6.0</TargetFramework>
    <ImplicitUsings>enable</ImplicitUsings>
    <Nullable>enable</Nullable>
  </PropertyGroup>
```

</Project>

And now within the `ScrapConsole/Program.cs` file, we can use our library

```
// See https://aka.ms/new-console-template for more information
using ScrapLibrary;

Console.WriteLine("Hello, World!");

ScrapLibrary.Class1.Hello();
```

Summary

I think you get the general idea here :) In summary, here are all of the commands we ran to create this project -

```
mkdir dotnet-scrap
cd dotnet-scrap
dotnet new sln
dotnet new classlib -o ScrapLibrary
dotnet sln add ScrapLibrary/ScrapLibrary.csproj
dotnet new xunit -o ScrapLibrary.Test
dotnet sln add ScrapLibrary.Test/ScrapLibrary.Test.csproj
cd ScrapLibrary.Test/
dotnet add reference ../ScrapLibrary/ScrapLibrary.csproj
cd ..
dotnet new console -o ScrapConsole
dotnet sln add ScrapConsole/ScrapConsole.csproj
cd ScrapConsole/
dotnet add reference ../ScrapLibrary/ScrapLibrary.csproj
# Build and run your console application! :D
dotnet build
dotnet run
Hello, World!
Hello!
cd ..
```

These commands produce the following project structure

```
tree -L 2
```

```
.
├─ dotnet-scrap.sln
├─ ScrapConsole
│   └─ bin
│   └─ obj
│   └─ Program.cs
│   └─ ScrapConsole.csproj
├─ ScrapLibrary
│   └─ bin
│   └─ Class1.cs
│   └─ obj
│   └─ ScrapLibrary.csproj
└─ ScrapLibrary.Test
    └─ bin
    └─ obj
    └─ ScrapLibrary.Test.csproj
    └─ UnitTest1.cs
```

9 directories, 7 files

Help / Documentation

The dotnet command contains very nice output for help and documentation. It is worth taking a few minutes to explore these options, as they will help you navigate the CLI effectively.

If you want basic help for a command to be output to your console

```
dotnet add --help
```

Description:

.NET Add Command

Usage:

```
dotnet [options] add [<PROJECT>] [command]
```

Arguments:

<PROJECT> The project file to operate on. If a file is not specified, the command will search the current directory for one. [default: /home/kapper/Code/dotnet-scrap/]

Options:

-, -h, --help Show command line help.

Commands:

package <PACKAGE_NAME> Add a NuGet package reference to the project.

reference <PROJECT_PATH> Add a project-to-project reference to the project.

If you have internet connection, the following command will open your default web browser and navigate to the documentation for the `dotnet add` command. The online documentation is much more verbose and contains several examples. We don't need to bookmark it or search it up each time though, because this command will take us right to the relevant page!

```
dotnet help add
```

There are many other help options worth exploring that the CLI provides. For example, the `dotnet new --list` command will output all available project templates that you can use

```
dotnet new --list
```

These templates matched your input:

Template Name	Short Name	Language	Tags
ASP.NET Core Empty	web	[C#],F#	Web/Empty
ASP.NET Core gRPC Service	grpc	[C#]	Web/gRPC
ASP.NET Core Web API	webapi	[C#],F#	Web/WebAPI
ASP.NET Core Web App	webapp,razor	[C#]	Web/MVC/Razor Pages
ASP.NET Core Web App (Model-View-Controller)	mvc	[C#],F#	Web/MVC
ASP.NET Core with Angular	angular	[C#]	Web/MVC/SPA
ASP.NET Core with React.js	react	[C#]	Web/MVC/SPA
ASP.NET Core with React.js and Redux	reactredux	[C#]	Web/MVC/SPA
Blazor Server App	blazorserver	[C#]	Web/Blazor
Blazor WebAssembly App	blazorwasm	[C#]	Web/Blazor/WebAssembly/PWA
Class Library	classlib	[C#],F#,VB	Common/Library
Console App	console	[C#],F#,VB	Common/Console
dotnet gitignore file	gitignore		Config
Dotnet local tool manifest file	tool-manifest		Config
EditorConfig file	editorconfig		Config
global.json file	globaljson		Config
MSTest Test Project	mstest	[C#],F#,VB	Test/MSTest
MVC ViewImports	viewimports	[C#]	Web/ASP.NET
MVC ViewStart	viewstart	[C#]	Web/ASP.NET
NuGet Config	nugetconfig		Config
NUnit 3 Test Item	nunit-test	[C#],F#,VB	Test/NUnit

NUnit 3 Test Project	nunit	[C#],F#,VB	Test/NUnit
Protocol Buffer File	proto		Web/gRPC
Razor Class Library	razorclasslib	[C#]	Web/Razor/Library
Razor Component	razorcomponent	[C#]	Web/ASP.NET
Razor Page	page	[C#]	Web/ASP.NET
Solution File	sln		Solution
Web Config	webconfig		Config
Worker Service	worker	[C#],F#	Common/Worker/Web
xUnit Test Project	xunit	[C#],F#,VB	Test/xUnit

This works for all commands available for the dotnet CLI. Without specifying a command, we will get the following output to our console.

```
dotnet help
.NET SDK (6.0.202)
Usage: dotnet [runtime-options] [path-to-application] [arguments]

Execute a .NET application.

runtime-options:
--additionalprobingpath <path>  Path containing probing policy and assemblies to probe for.
--additional-deps <path>        Path to additional deps.json file.
--depsfile                      Path to <application>.deps.json file.
--fx-version <version>         Version of the installed Shared Framework to use to run the application.
--roll-forward <setting>       Roll forward to framework version (LatestPatch, Minor, LatestMinor, Major,
LatestMajor, Disable).
--runtimeconfig                 Path to <application>.runtimeconfig.json file.

path-to-application:
The path to an application .dll file to execute.

Usage: dotnet [sdk-options] [command] [command-options] [arguments]

Execute a .NET SDK command.

sdk-options:
-d|--diagnostics  Enable diagnostic output.
-h|--help        Show command line help.
--info           Display .NET information.
--list-runtimes  Display the installed runtimes.
--list-sdks      Display the installed SDKs.
```

--version Display .NET SDK version in use.

SDK commands:

add	Add a package or reference to a .NET project.
build	Build a .NET project.
build-server	Interact with servers started by a build.
clean	Clean build outputs of a .NET project.
format	Apply style preferences to a project or solution.
help	Show command line help.
list	List project references of a .NET project.
msbuild	Run Microsoft Build Engine (MSBuild) commands.
new	Create a new .NET project or file.
nuget	Provides additional NuGet commands.
pack	Create a NuGet package.
publish	Publish a .NET project for deployment.
remove	Remove a package or reference from a .NET project.
restore	Restore dependencies specified in a .NET project.
run	Build and run a .NET project output.
sdk	Manage .NET SDK installation.
sln	Modify Visual Studio solution files.
store	Store the specified assemblies in the runtime package store.
test	Run unit tests using the test runner specified in a .NET project.
tool	Install or manage tools that extend the .NET experience.
vstest	Run Microsoft Test Engine (VSTest) commands.
workload	Manage optional workloads.

Additional commands from bundled tools:

dev-certs	Create and manage development certificates.
fsi	Start F# Interactive / execute F# scripts.
sql-cache	SQL Server cache command-line tools.
user-secrets	Manage development user secrets.
watch	Start a file watcher that runs a command when files change.

Run 'dotnet [command] --help' for more information on a command

Revision #5

Created 30 April 2022 12:00:21 by Shaun Reed

Updated 4 May 2022 18:26:21 by Shaun Reed