# .NET CLI dotnet SDK commands

**START HERE**

`dotnet new console -o <name>`
New console app

`dotnet add package <id>`
Add a NuGet package

`dotnet run`
Build and run

`dotnet watch`
Rerun on file change

`dotnet build`
Compile the project

`dotnet publish -c Release`
Make a deployable build

`dotnet test`
Run the tests

`dotnet new sln`
Start a solution

**Source** .cs + .csproj → `restore` → **Packages** obj/ project.assets.json → `build` → **Binaries** bin/<config>/<tfm> → `publish` → **Deploy folder** bin/.../publish

## CONCEPTS

Build, run, test and publish restore NuGet packages first, so a separate restore is rarely needed.

Without a path argument, commands pick the one project or solution file in the current folder; with several, name one.

Arguments the app should get go after a literal --: dotnet run -- --help.

MSBuild properties pass through with -p:Name=Value, so most project settings can be set per call.

## CREATE

`dotnet new list`
List installed templates

`dotnet new console -o <name>`
New console app in a folder

`dotnet new classlib -o <name>`
New class library

`dotnet new webapi -o <name>`
New ASP.NET Core web API

`dotnet new xunit -o <name>`
New xUnit test project

`dotnet new sln`
New solution file
*Creates .slnx on .NET 10; .NET 9 and earlier create .sln.*

`dotnet new gitignore`
Add a .gitignore for .NET

`dotnet new globaljson --sdk-version <v>`
Pin the SDK version

`dotnet new tool-manifest`
Create .config/dotnet-tools.json

`dotnet new search <text>`
Find templates on NuGet.org

`dotnet new install <package>`
Install a template pack

`dotnet new <template> -f <tfm>`
Choose the target framework

## BUILD AND RUN

`dotnet restore`
Restore NuGet packages

`dotnet build`
Compile the project or solution

## BUILD AND RUN (cont.)

`dotnet build -c Release`
Build in Release configuration

`dotnet build --no-restore`
Skip the implicit restore

`dotnet build --no-incremental`
Force a full rebuild

`dotnet build -f <tfm>`
Build one target framework

`dotnet build -o <dir>`
Write binaries to a folder
*Not allowed on a whole solution since .NET 7.0.200.*

`dotnet build -v <level>`
Set output verbosity

`dotnet build -bl`
Write msbuild.binlog

`dotnet clean`
Delete build outputs

`dotnet run`
Build and run the project

`dotnet run --project <path>`
Run a project from elsewhere

`dotnet run -c Release`
Run the Release build

`dotnet run -- <args>`
Pass arguments to the app

`dotnet run -lp <name>`
Use a launch profile

`dotnet run app.cs`
Run a single-file C# app
*File-based apps need SDK 10 or later.*

`dotnet <app>.dll`
Run a built framework-dependent app

`dotnet watch`
Rerun or hot reload on change

`dotnet watch test`
Rerun tests on change

## TEST

`dotnet test`
Build and run all tests

`dotnet test --filter <expr>`
Run only matching tests
*Operators = != ~ !~, joined by | (or) and & (and); a bare word means FullyQualifiedName~word.*

`dotnet test --filter "Name~<text>"`
Match by test name

`dotnet test --filter "Category=<name>"`
Match by category or trait

`dotnet test --no-build`
Skip building first

## TEST (cont.)

`dotnet test -t`
List tests without running

`dotnet test -l "console;verbosity=detailed"`
Show each test result

`dotnet test -l trx`
Write a .trx results file

`dotnet test --collect "XPlat Code Coverage"`
Collect coverage with Coverlet
*Needs the coverlet.collector package in the test project.*

`dotnet test --blame`
Find the test that crashed the host

## PACKAGES AND REFERENCES

`dotnet add package <id>`
Add a NuGet package
*From .NET 10 also dotnet package add <id>.*

`dotnet add package <id> -v <version>`
Add a specific version

`dotnet remove package <id>`
Remove a package

`dotnet list package`
List referenced packages

`dotnet list package --outdated`
Show newer versions

`dotnet list package --vulnerable`
Show known vulnerabilities

`dotnet add reference <csproj>`
Reference another project

`dotnet pack -c Release`
Create a .nupkg

## SOLUTIONS

`dotnet sln add <csproj>`
Add a project to the solution

## PUBLISH

`dotnet publish -c Release`
Publish for deployment
*Publish already defaults to Release for net8.0 and later projects.*

`dotnet publish -o <dir>`
Publish into a folder

`dotnet publish -r <rid> --self-contained`
Bundle the runtime

`dotnet publish -p:PublishSingleFile=true`
One-file executable

## TOOLS AND FORMAT

`dotnet tool install -g <id>`
Install a global tool

`dotnet tool install <id>`
Install a local (repo) tool
*Recorded in .config/dotnet-tools.json; SDK 10 creates the manifest on demand.*

`dotnet tool restore`
Install tools from the manifest

`dotnet tool run <name>`
Run a local tool

`dotnet format`
Apply .editorconfig style fixes

`dotnet format --verify-no-changes`
CI check, fails if changes needed

## SDK AND WORKLOADS

`dotnet --info`
SDK, runtime and OS details

`dotnet --version`
SDK version in use

`dotnet --list-sdks`
Installed SDKs

`dotnet --list-runtimes`
Installed runtimes

`dotnet <command> -h`
Help for any command

global.json in a parent folder pins the SDK version dotnet picks; otherwise the newest installed SDK is used.

## RECIPES

**New app with tests in a solution**

```
dotnet new sln -n App
dotnet new console -o src/App
dotnet new xunit -o tests/App.Tests
dotnet sln add src/App tests/App.Tests
dotnet add tests/App.Tests reference src/App/App.csproj
dotnet test
```

**CI pipeline steps**

```
dotnet restore
dotnet format --verify-no-changes --no-restore
dotnet build -c Release --no-restore
dotnet test -c Release --no-build
```

**Single-file Linux binary**

```
dotnet publish -c Release -r linux-x64 \
  --self-contained -p:PublishSingleFile=true \
  -o out
```