The Code Whisperer

Your code is trying to tell you something.

- Ask Me a Question
- Past Articles
- Integrated Tests are a Scam
- On-Site Training
- Online Training
- JUnit on One Page

- Twitter
- GitHub
- Stack Overflow
- RSS

From Zero To NUnit With Visual Studio Code

July 21, 2018 Tutorials Comments

I have a new Linux laptop and I wanted to run C# code. I had no idea where to start. I last wrote C# for money in 2004. It took me over an hour of hunting to figure out how to run a single test, so I decided to write a tutorial that could help someone else go from zero to NUnit with Visual Studio Code.

Please add comments with suggestions for clearer explanations, additional useful details, or better instructions.

I'm especially interested in knowing how to package these setup instructions to make them repeatable, since I don't know enough yet about *how to do* infrastructure as code.

Assumptions

For this tutorial, I assume that you know nothing about the modern .NET environment. You don't know how to install anything, where to find it, nor how to use it. You might know the following:

- I think I need to use Visual Studio or something like it. I've heard of VS Code.
- I know I want to write tests with NUnit.
- I know that I need to add assembly references to my C# project in order to use NUnit.

You've never installed any of this stuff before—or, at least, you've never knowingly nor intentionally installed it.

Environment

I'm running Pop!_OS on a System 76 Galago Pro laptop. You can safely think of this as Ubuntu 18.04. To my knowledge, I have nothing about .NET installed at all.

I use `etckeeper` in order to be able to roll back the contents of `/etc` on my file system. I use `stow` in order to be able to roll back the contents of my home directory's dotfiles.

The Goal

I have one immediate goal: I have some C# code with NUnit tests and I want to be able to run those tests. The C# code consists of two classes and has no entry point ("main"). I judge success by being able to run these tests as see the test results.

Instructions

I wanted to install VS Code, create a project, paste in some C# code, and then run the tests. I tried to do exactly that, it didn't work, and I gathered enough information to figure out how to make it work. Now I have reordered those instructions to make them appear more orderly and intentional.

Summary

- Install the .NET framework.
- Install Visual Studio Code.
- Create a C# project in VS Code.
- Add some packages to the project related to testing.
- Remove the entry point class that VS Code generated.
- Build and run the tests.

Details

First, verify that you don't have anything installed related to .NET nor VS Code.

```
$ dotnet
[Command not found]
$ code
[Command not found]
[Try to run "Code" from the desktop manager's application launcher. Nothing.]
```

Now, install the .NET framework. Search the web for `install net core sdk ubuntu`. Find https://www.microsoft.com/net/download/linux-package-manager/ubuntu18-04/sdk-current. Follow its instructions.

Install .NET Core SDK on Linux Ubuntu 18.04

First, I need to install the Microsoft key, their product repository, and required dependencies.

```
$ sudo etckeeper unclean
[Nothing, so /etc has no uncommitted changes.]
# I just like to download things here.
$ pushd $HOME/Downloads
$ wget -q https://packages.microsoft.com/config/ubuntu/18.04/packages-microsoft-prod.deb
$ ls
[Yep, I see the new file.]
$ sudo dpkg -i packages-microsoft-prod.deb
$ popd
$ pushd /etc; sudo git status -s
[New files in /etc/apt that need committing.]
$ popd
$ sudo etckeeper commit "We can now install products from Microsoft's repository"
# I guess I need this to use package repositories over HTTPS.
$ sudo apt-get install apt-transport-https
# Now I can safely use the Microsoft package repository.
$ sudo apt-get update
```

I still don't quite trust `etckeeper`, so I check the state of `/etc`.

```
$ pushd /etc; sudo git status -s
[No uncommitted changes.]
$ sudo git log
[Evidence of having saved uncommitted changes, although that might have come from a previous working session. Either way, it's fine.]
$ popd
```

Finally, I can install .NET Core SDK 2.1.

```
$ sudo apt-get install dotnet-sdk-2.1
[Evidence of installing a bunch of stuff, almost certainly related to .NET Code SDK 2.1]
[Evidence of etckeeper committing after the installation.]
# Verify
$ dotnet
[The dotnet usage message.]
```

Install Visual Studio Code

Since I have Microsoft's product repository, I assume that it contains a version of Visual Studio Code.

```
$ sudo apt-cache search code
["Code Editing. Redefined." That's marketing!]
$ sudo apt-get install code
[Evidence of installation.]
[Evidence that etckeeper committed after the installation.]
```

Next, I launch the newly-installed application named "Code". It launches, then "helpfully" opens a web page telling me how to get started, forcing me to put focus back on the actual application that I launched.

Opt Out of Telemetry Reporting

So before I have the faintest idea how to use VS Code to do anything, a helpful window pops up telling me that I'm going to send "telemetry information" to Microsoft as I use VS Code. No, thanks.

Now I have to learn about putting VS Code settings under version control, because I know that configuration settings in tools that I don't know well can easily spiral out of control in minutes, and then I waste hours trying to get back on track.

Thank you, Microsoft.

Fortunately, this doesn't hurt too badly. First, I go to `$HOME/.config` to find a new directory `Code`. This seems to contain some combination of configuration data and temporary storage for VS Code. I want to put *some* of this stuff in version control, but not *all*, because programmers don't care enough about separating configuration settings (I want in version control) from temporary storage (I definitely don't want in version control).

Since I use `stow` for my dotfiles, and I had previously stowed my `.config` directory, `$HOME/.config` really points to `$DOTFILES/common/config/.config`, where `DOTFILES` is the root of a git repository and my "all my dotfiles" project. That git repository ignores `common/config/.config` in general, because of so many high-churn files, so I have to choose carefully which files inside `.config` I want to track in that git repository.

I dig inside `$DOTFILES/common/config/.config/Code` and find user settings file called `User/settings.json`. I want to keep versions of this.

```
$ cd $DOTFILES
$ git add --force common/config/.config/Code/User/settings.json
$ git commit -m "We now track changes to Microsoft Visual Studio Code user-level configuration settings."
```

I learned later that `User/settings.json` doesn't exist until you add a user-level preference setting to override the system-level preferences. Great!

In VS Code, I choose **File > Preferences > Settings**. This opens an editor window on an empty JSON document describing your user-level preferences. **I save this file before changing it.** This last action creates the file `User/settings.json`, which I commit to my dotfiles repository before I start changing things and fall into a ditch somewhere.

Now I can follow the instructions at https://code.visualstudio.com/docs/supporting/faq#_how-to-disable-telemetry-reporting to disable sending telemetry information to Microsoft.

First, I add the preference setting to the user-level preference settings file, then I commit the change to my dotfiles repository, and then I restart VS Code.

I verify this step by noting that VS Code no longer pops up a warning about sending telemetry information and I simply hope that Microsoft doesn't lie to me. It'll be fine.

Install VS Code Extensions

Now I install the extensions I'll use in VS Code related to C# and NUnit.

I visit https://code.visualstudio.com/Docs/languages/csharp and that leads me to https://marketplace.visualstudio.com/items?itemName=ms-vscode.csharp. Inside VS Code, I execute `ext install ms-vscode.csharp` using the **Quick Open** command.

```
# Inside VS Code
Press Ctrl+P to launch VS Code Quick Open
Paste "ext install ms-vscode.csharp" into the little dialog.
[A window pops up to show evidence that there is now a C# extension installed.]
```

Create The VS Code Project

Since tools like VS Code always have magical wizards to create new projects and since I don't know much about the contents of a C# project, I ask VS Code to create a new project, so that I can delete most of it and replace it with the code I want to run.

And… of *course*, VS Code doesn't have a menu item for "new project", so I hunt down how to do that. Happily, the .NET SDK makes that relatively painless and I *like* doing this from the transparency of the command line.

```
$ cd $WORKSPACES_ROOT  # The path where I like to put all my projects.
$ dotnet new console -o $PROJECT_NAME  # PROJECT_NAME is a directory name
[Evidence of creating a project.]
# Verify
$ find $PROJECT_NAME
[Program.cs and some object files and a project file of type .csproj.]
```

Next, I put these assets into version control, but again, I don't know what I'm doing, so I rely on gitignore.io to generate `.gitignore` for me.

```
$ cd $PROJECT_NAME
$ git init
$ curl -o .gitignore https://www.gitignore.io/api/csharp,vscode
$ cat .gitignore
[It seems to have a lot of useful things in it.]
$ git add --dry-run --all
[This only adds .gitignore, Program.cs, and the project file, which seems perfectly reasonable so far.]
$ git add --all
$ git commit --message "We now have the default .NET C# project."
```

I open this project in VS Code by opening Explorer (**View > Explorer**), pressing **Open Folder**, then choosing `$PROJECT_NAME`, meaning the root of the project that I created with `dotnet new`.

Right away, VS Code tries to build the project, and it pops up a window telling me that the project needs some more assets to build the project, so I agree to add those assets by pressing **Yes**. Since that adds things that `.gitignore` appears not to ignore, I look at them to decide for myself whether I want to track them in version control.

The file `.vscode/launch.json` appears to have launch commands and that seems handy to track. The file `.vscode/tasks.json` appears to have a build command and that seems handy to track. I decide to track them both.

```
$ git add --all
$ git commit --message "We can now build the project in VS Code."
```

Build And Run The Project Without VS Code

Since .NET SDK offers these nice commands, I want to learn to use them.

```
$ cd $PROJECT_ROOT   # Currently $WORKSPACE_ROOT/$PROJECT_NAME
$ dotnet build
[Evidence of building. No warnings, no errors.]
$ dotnet run
Hello World!
# Program.cs indeed asks to print this message to the console, so everything seems to work.
```

Build And Run The Project With VS Code

I go back inside VS Code and choose **Tasks > Run Build Task**, then enter **build** for the name of the task. VS Code shows evidence of building the project. This task seems to correspond with the entry in `.vscode/tasks.json`, so I learn something there.

Next, I choose **Debug > Start Without Debugging**.

The UX specialists at Microsoft decided that **Run** would confuse programmers, so instead they chose to name the menu item **Start Without Debugging** and put it under the menu item **Debug**.

This tells me a lot about what the average .NET programmer expects. *Not* debugging becomes the exception, rather than the rule.

This also reminds me of `sudo add-apt-repository -remove {repository_name}`. Srsly.

I see `Hello World!` in the **Debug Console**, so now I can run this project from both the command line and with a single keystroke (`Ctrl+F5`).

Run Tests With NUnit

This step requires some extra work and I had to hunt down the details. I naively installed something and didn't know that it depended on something else and so on. I hope that this saves you some headache.

```
$ cd $PROJECT_ROOT
$ dotnet add package Microsoft.NET.Test.Sdk
$ dotnet add package Nunit3TestAdapter
$ dotnet add package NUnit
$ git add --all
$ git commit --message "We can now write and run tests with NUnit."
```

By installing all these packages into the project, I could run tests, at least from the command line. Almost.

```
$ dotnet test
Program.cs(7,21): error CS0017: Program has more than one entry point defined. Compile with /main to specify the type that contains the entry point. [$PROJECT_ROOT/$PROJECT_NAME.csproj]
```

I *believe* that I only have one entry point, so I tried for a few minutes to figure out this message, but I gave up due to not finding anything simple and straightforward to read. Instead, I now simply remove `Program.cs` and replace it with the code that I want to run, which contains tests.

After doing that, I can run the tests.

```
$ dotnet test
Build started, please wait...
Build completed.

Test run for $PROJECT_PATH/bin/Debug/netcoreapp2.1/$PROJECT_NAME.dll(.NETCoreApp,Version=v2.1)
Microsoft (R) Test Execution Command Line Tool Version 15.7.0
Copyright (c) Microsoft Corporation.  All rights reserved.

Starting test execution, please wait...
Failed   $FAILING_TEST_NAME
Error Message:
 System.ArgumentOutOfRangeException : Index was out of range. Must be non-negative and less than the size of the collection.
Parameter name: index
Stack Trace:
   at System.Collections.Generic.List`1.get_Item(Int32 index)
   [...]

Total tests: 9. Passed: 8. Failed: 1. Skipped: 0.
Test Run Failed.
Test execution time: 0.7209 Seconds
```

I can even run the tests in VS Code. I open the file containing the tests, wait a moment, and see **Run All Tests** as an option beneath the `[TestFixture]` annotation where I declare the test class. When I choose this option, VS

Code runs the tests and reports the same results as `dotnet test`.

With this, I feel satisfied. I can run tests.

Removing It All

In order to remove all this stuff, this seems to suffice. First, I close VS Code. After that I issue these commands.

```
# Remove the project
$ rm -fr $PROJECT_ROOT
# Remove the VS Code user-level settings
$ rm -fr $DOTFILES/common/config/.config/Code
# Remove VS Code
$ sudo apt-get remove code
# Remove .NET SDK
$ sudo apt-get remove dotnet-sdk-2.1
# I probably want to keep this installed, but I don't have enough experience to judge. I welcome your opinions.
$ sudo apt-get remove apt-transport-https
# I certainly want to remove the rest of this stuff!
$ sudo dpkg --remove packages-microsoft-prod
# I didn't expect to need to do this. See "By The Way..." below.
$ sudo dpkg --purge packages-microsoft-prod
$ sudo apt-get update
$ sudo etckeeper commit "Removed .NET SDK, C#, and VS Code."
```

Future Work

I'm not planning to work in C# on VS Code in a professional setting, so this sufficed for me. Even so, I can imagine wanting to add at least the following.

- Run this entire setup in a container of some kind to isolate it from the rest of the system. Docker?
- Somebody must have built some kind of decent GUI test runner that runs inside VS Code to run these tests. What plays the role of Test-Driven .NET for VS Code?
- How do I organize the files in a C#/NUnit project? I presume that I want separate assemblies for production code and test code, so that I can ship the first without the second.

If you can write or point to a tutorial that helps answer these questions, then please do. I would happily expand this tutorial to include information like that.

By The Way…

I learned something about `dpkg` the hard way while preparing this tutorial: I don't trust the word `remove` in `dpkg -remove` to mean "remove", because it doesn't necessarily remove things, at least according to my understanding of "remove".

```
# I need to remove Microsoft's package repository so that I can reinstall it for this tutorial.
$ sudo dpkg --remove packages-microsoft-prod
$ sudo rm /etc/apt/sources.list.d/microsoft-prod.list
# Now reinstall!
$ sudo dpkg --install $DOWNLOADS/packages-microsoft-prod.deb
$ sudo apt-get update
# Wait... I don't see the Microsoft repository in the list. Hm.
$ sudo apt-get install dotnet-sdk-2.1
["I've never heard of this 'dotnet-sdk-2.1'."]
```

Strange. I expect `-remove` to remove things. I guess not. After about 15 minutes of searching the web without even knowing which specific keywords would help me—I *love* that—I stumble upon a clue examining the switches on `dpkg`. I find a switch called `-purge`.

Wait a moment! If `-remove` removes things, then why do you need `-purge`? This must mean that `-remove` only *sometimes* removes things or it removes things for some *weaker* meaning of "remove". I figured that it couldn't hurt to try.

```
$ sudo dpkg --purge packages-microsoft-prod
$ sudo dpkg --install $DOWNLOADS/packages-microsoft-prod.deb
$ sudo apt-get update
[Output that includes listing a repository with the name "microsoft" in it.]
```

If you remove something with `dpkg`, but it doesn't seem to completely disappear, then try purging it. If you understand this better, then please explain it in the comments so that I can improve this document.

Comments