Visual Studio Code Quick Guide for PSh Users

Abstract:

(jump to TOC)

This document is a quick guide for PowerShell users to help them get working with the Visual Studio Code editor (which supplants the deprecated PSh ISE editor).

Intended Audience:

(jump to TOC)

This document is high-level and is intended to be used by persons who are familiar with Windows, PowerShell scripting, and the Microsoft ISE editor.

Document Revision and History:

(jump to TOC)

version	date	description
1.0	2020.03.22.Sun.1833	document creation

Freeware License and Disclaimer:

(jump to TOC)

About the Author:

(jump to TOC)

Daniel L. Benway

Active Directory and Information Security Architect / Engineer
BSc CS, MCSE (NT4, 2000), MCTS (SCCM 2012), CISSP, Security+, Network+, CCNA (2.0), CLP (AD R4)

LinkedIn http://www.LinkedIn.com/in/DanielLBenway

Blog http://www.DanielLBenway.net

twitter @Daniel_L_Benway

Special Thanks:

(jump to TOC)

- Special thanks to Frank Jaroneski, and Brandon Shcreiber for helping me explore and troubleshoot my VSC installation.

Table of Contents:

Installation:

(jump to TOC)

Install:

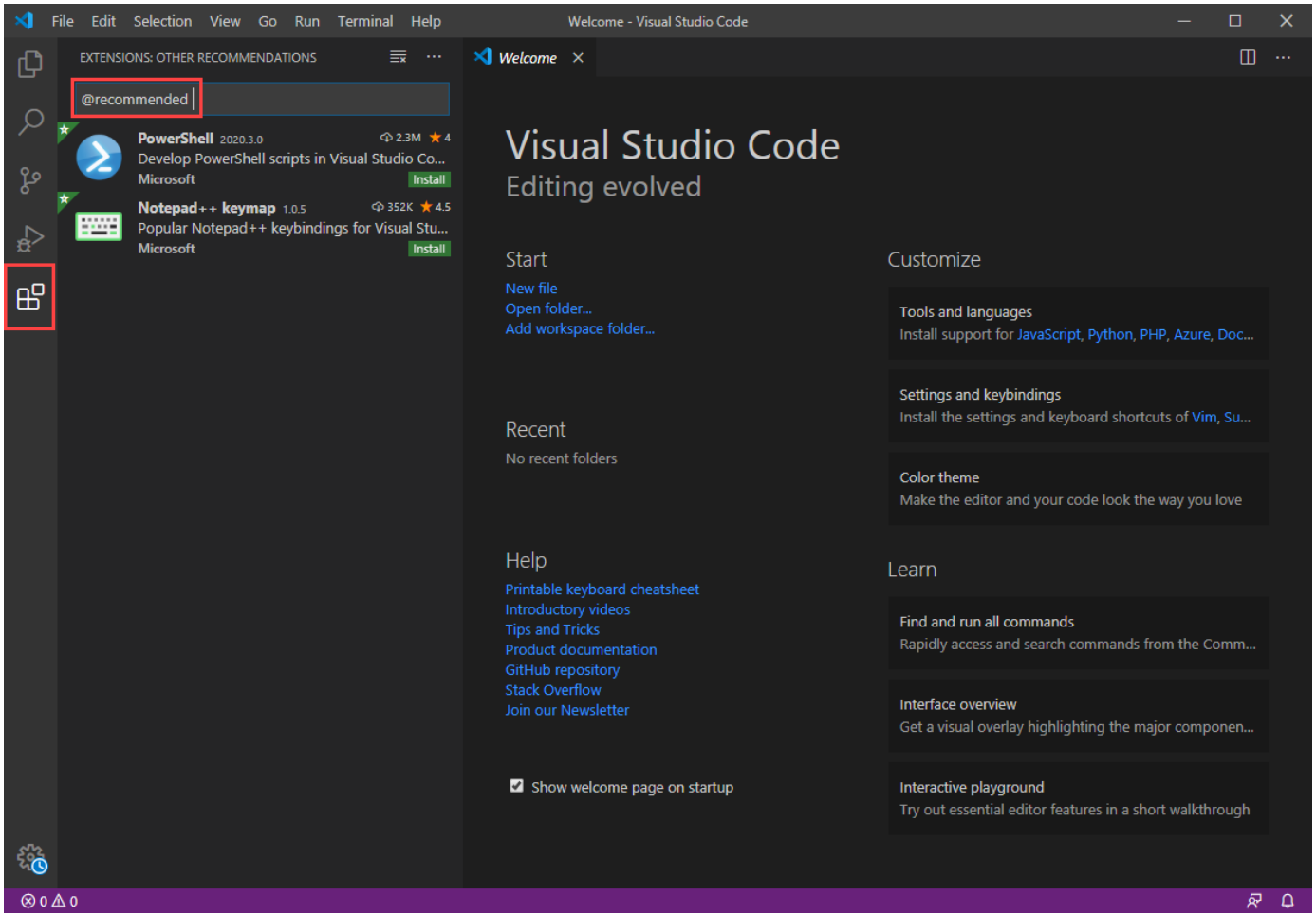
- Install PSh 5.1, and 7 (the most current versions of pre-core, and core) before installing Visual Studio Code, or else some components won't work properly.
- https://code.visualstudio.com/download

Configuration:

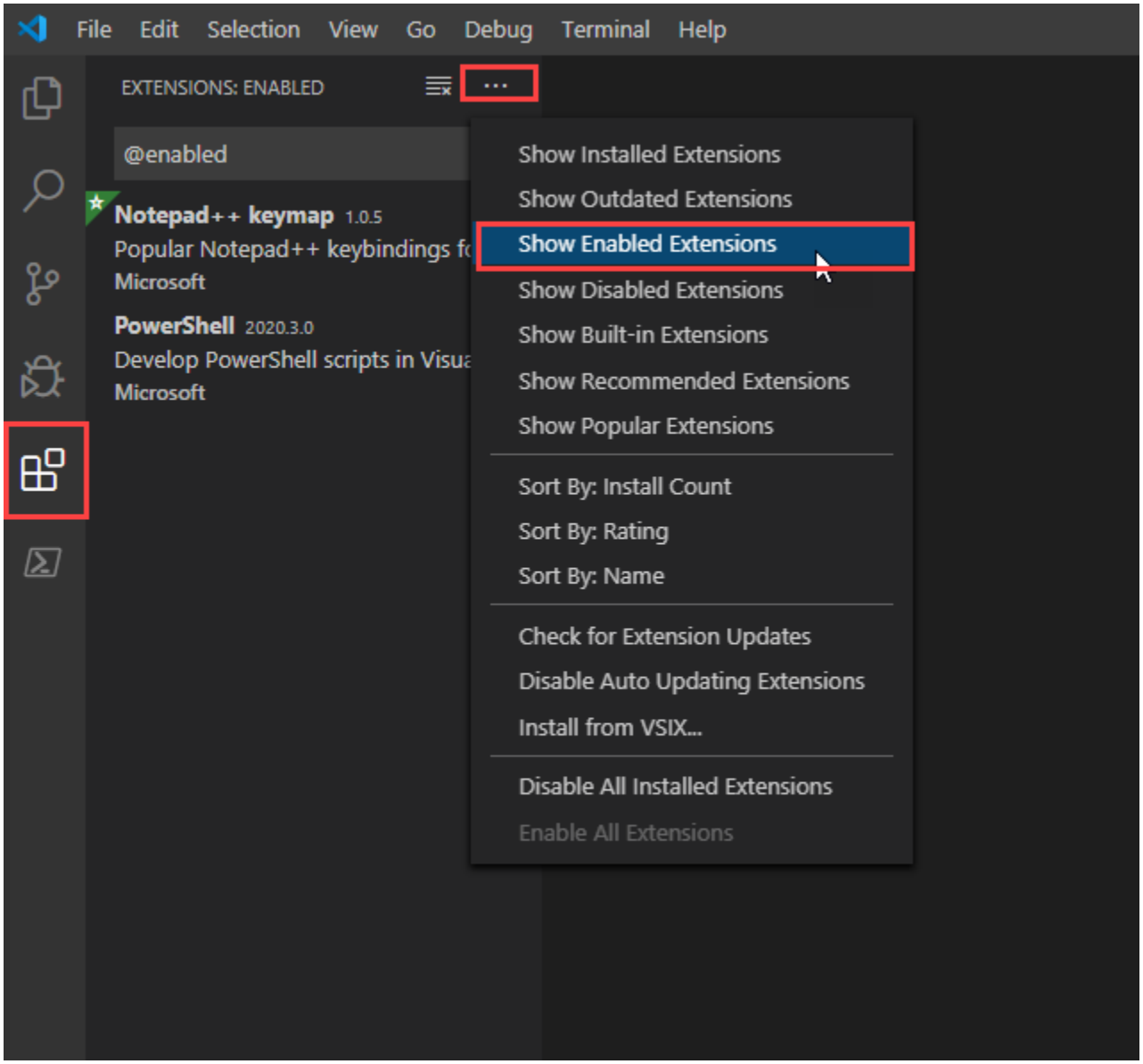
(jump to TOC)

Note: when you open Visual Studio Code as a regular user, vs. when you open it by 'run as Administrator', you have two separate environments, each with their own separate configurations

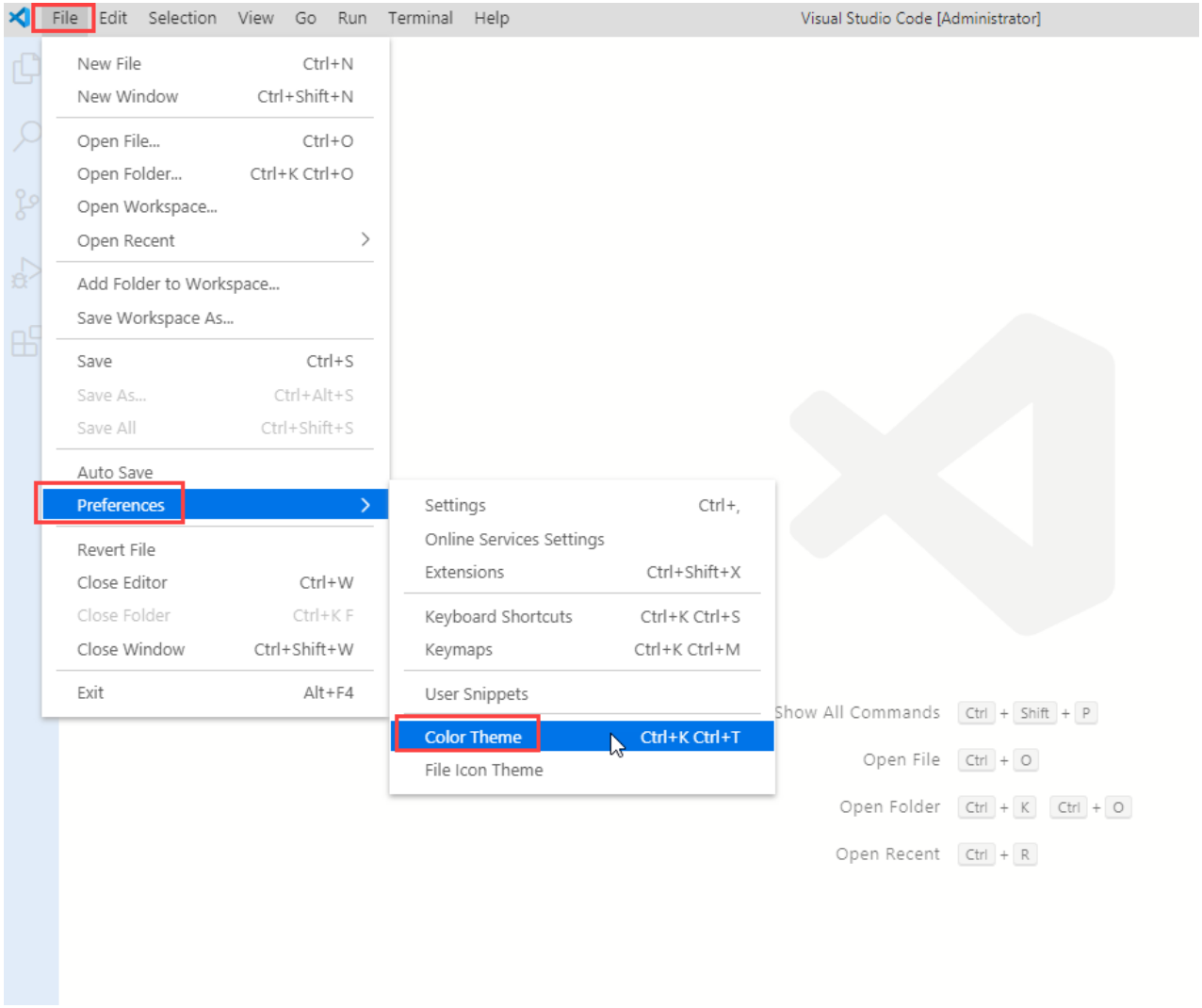
- add the recommended **PowerShell** extension (from Microsoft) and **NotePad++** (from Microsoft) extension:



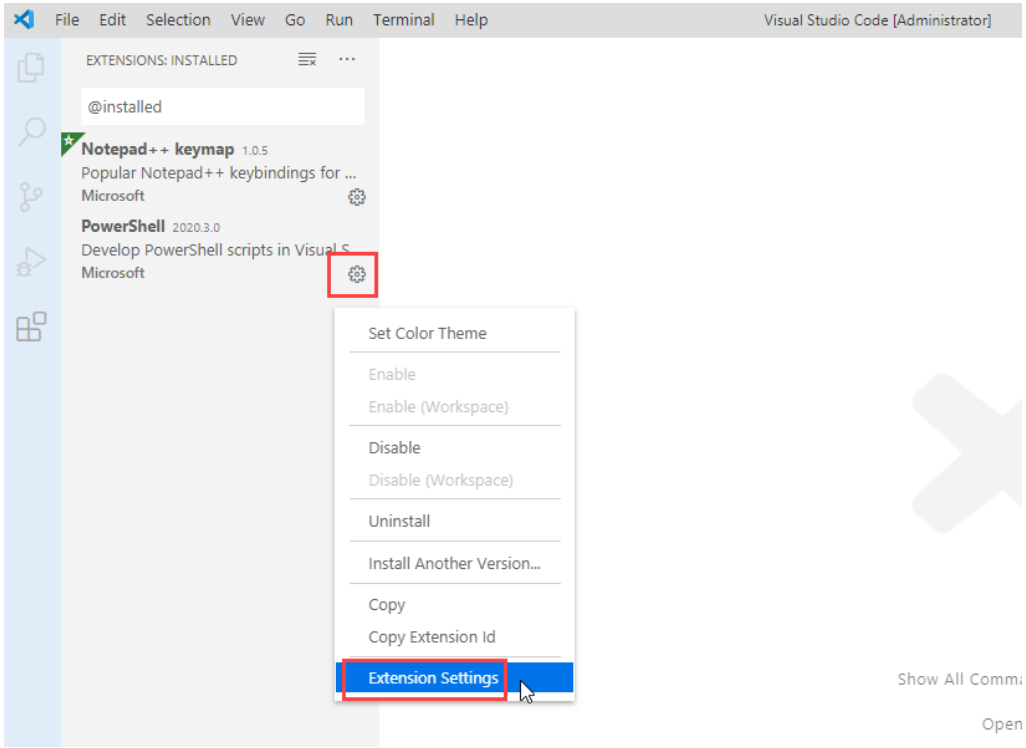
- Confirm enabled extensions:

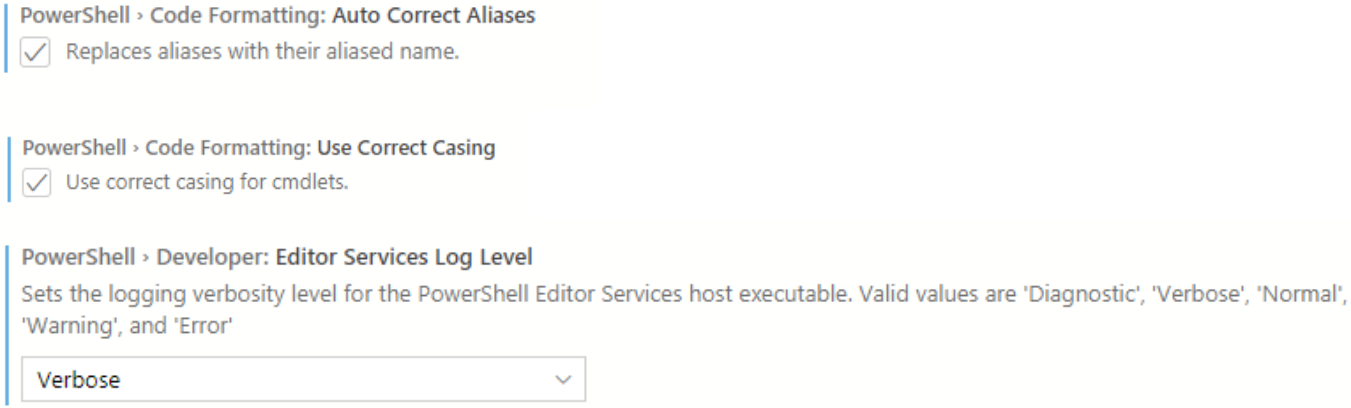


- change color theme to PowerShell ISE (File > Preferences > Color Theme):

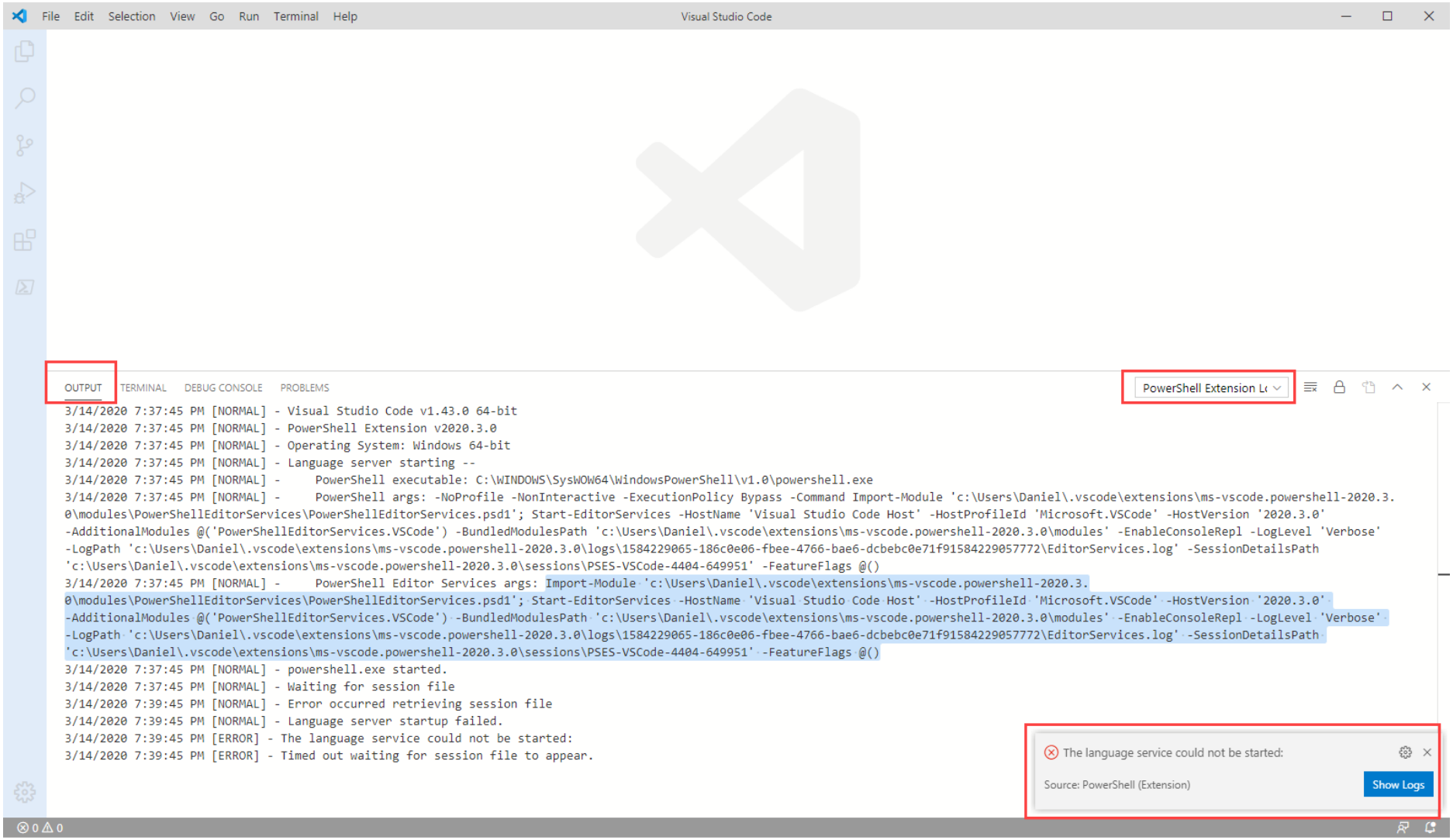


- Configure the PSh extension:





- Troubleshooting step: Take any error message out of the PowerShell Extension Output and run from a PSh prompt or ISE, and then Google the ensuing error message:



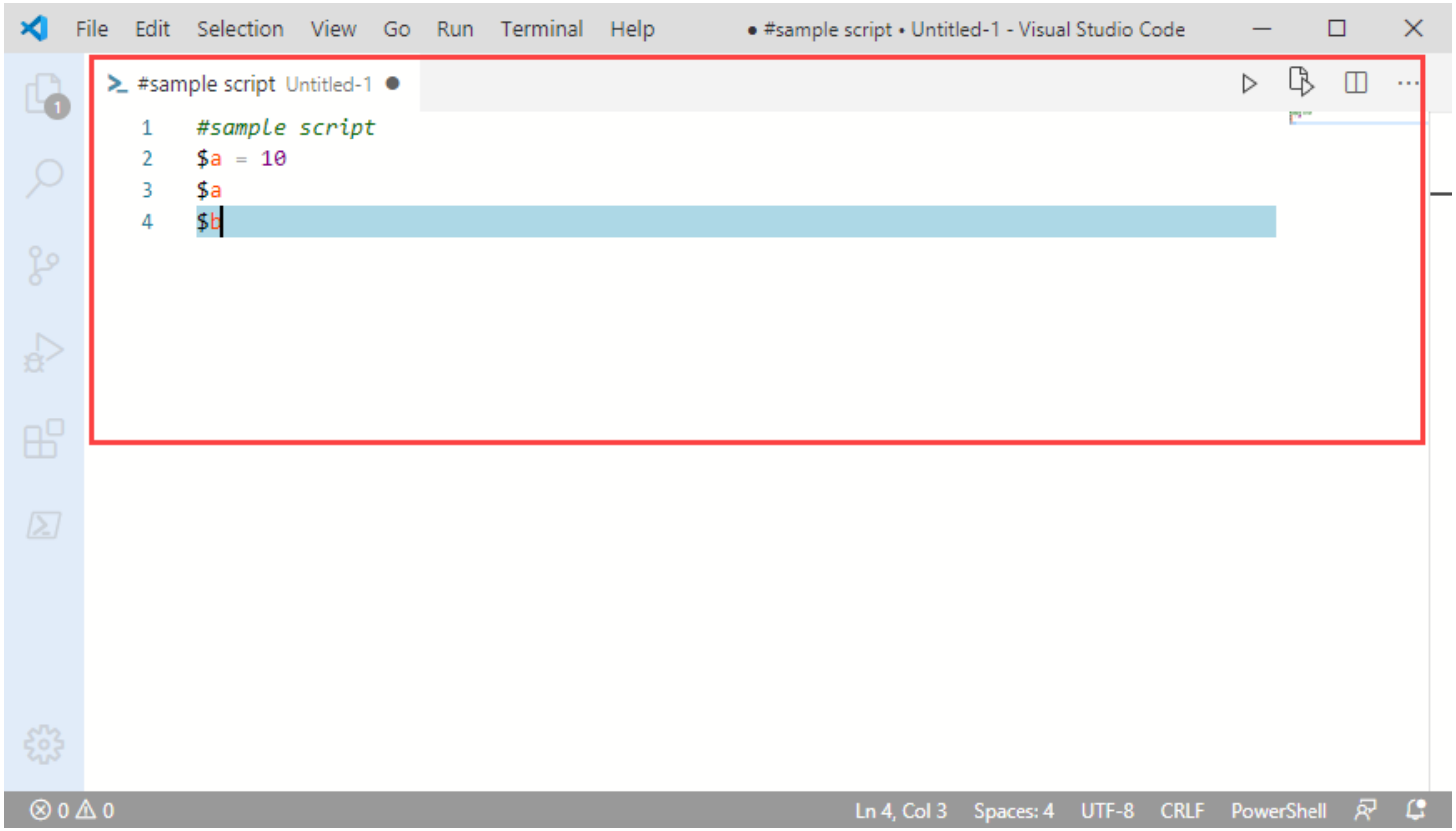
Usage:

(jump to TOC)

Just like ISE, launch Visual Studio Code as Administrator when you need those privileges.

Top Window:

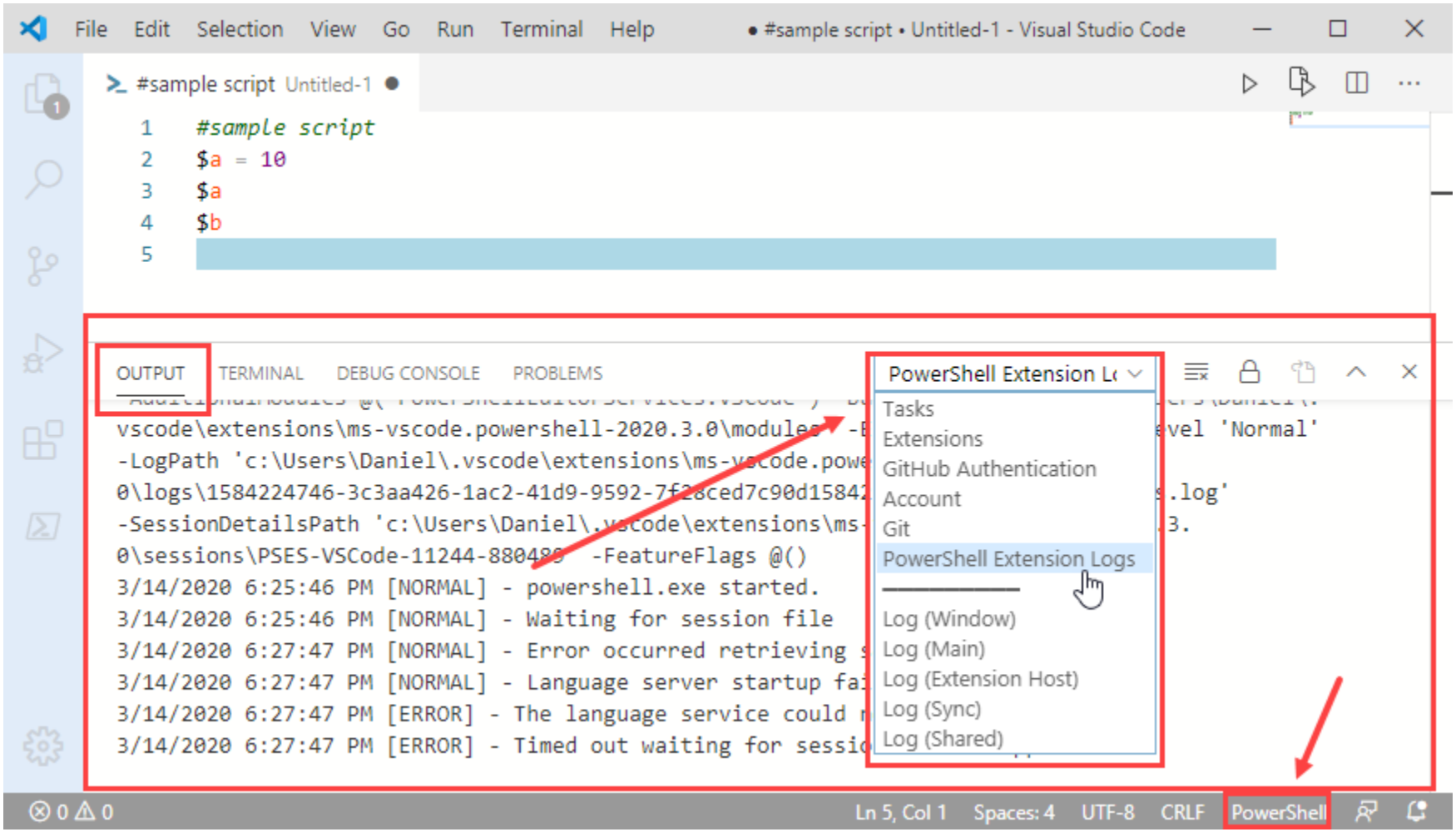
The top window is the 'file' window (where you write your PSh code):



Bottom Window:

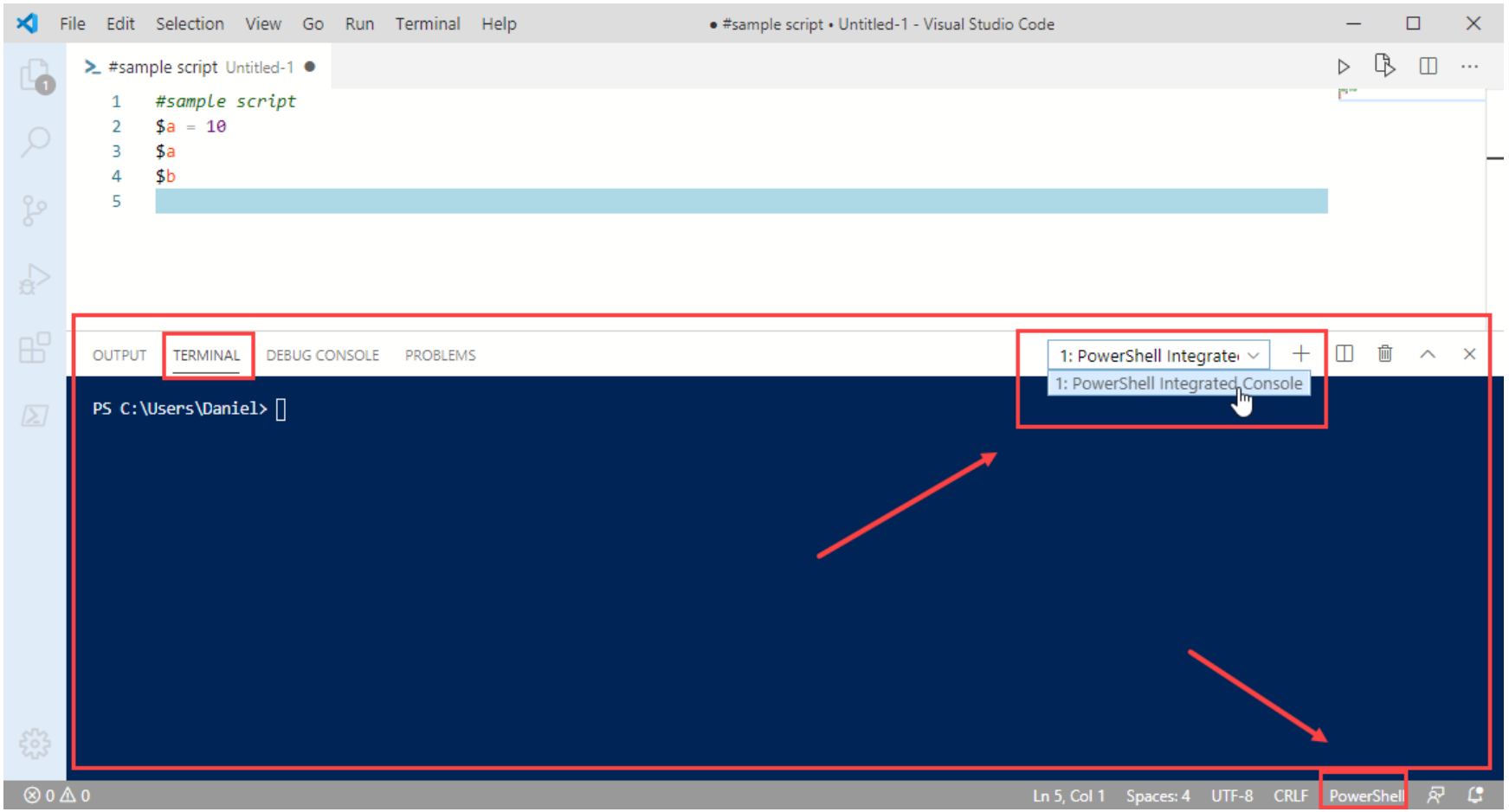
The bottom window has sub-windows: output, terminal, debug console, and problems.

Be mindful of the scope of the 'output' sub-window, as well as the language:



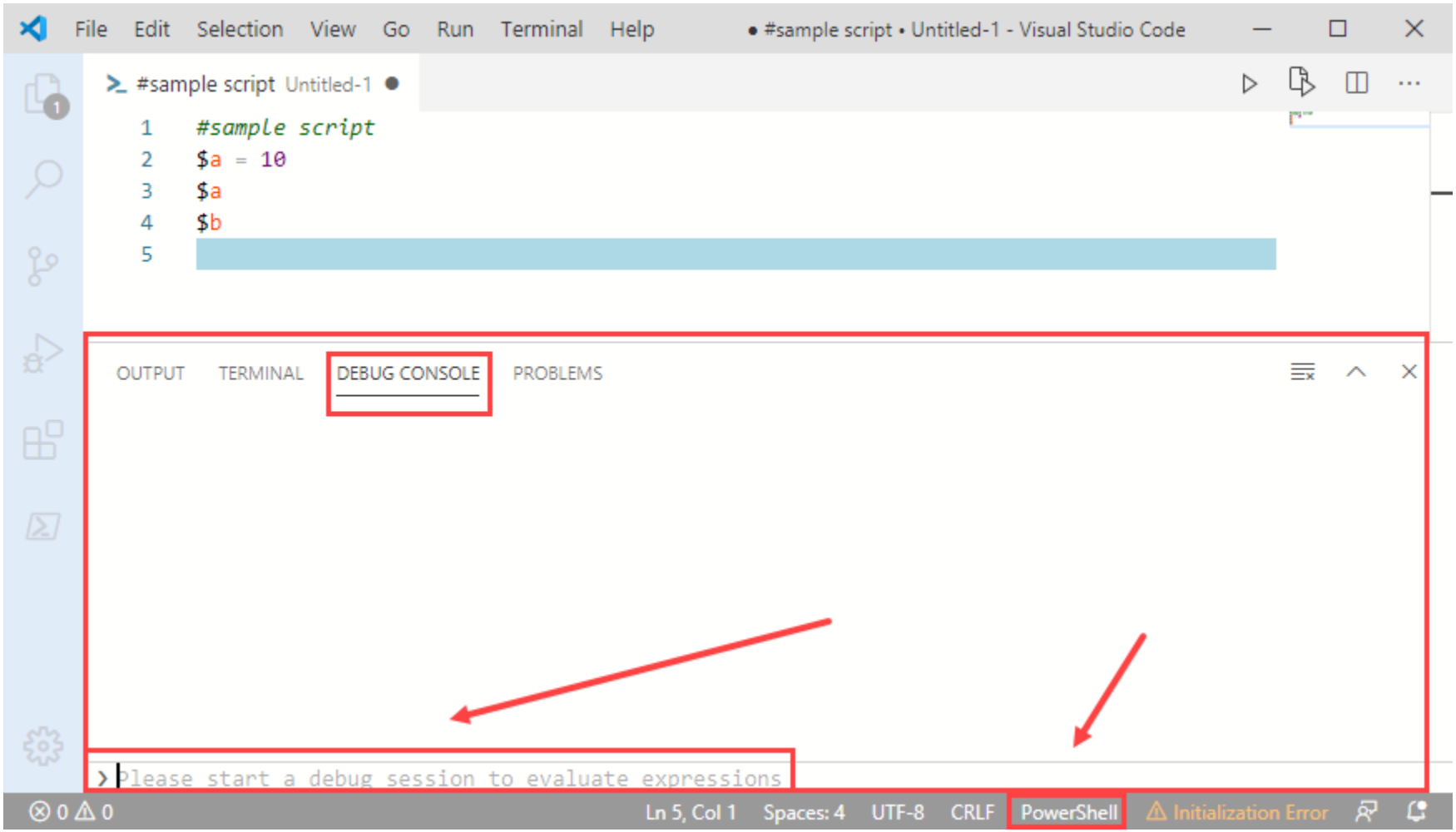
NOTE: the bottom window is loosely tied* to the top window (explained later)

Be mindful of the scope of the 'terminal' sub-window (explained later), as well as the language:



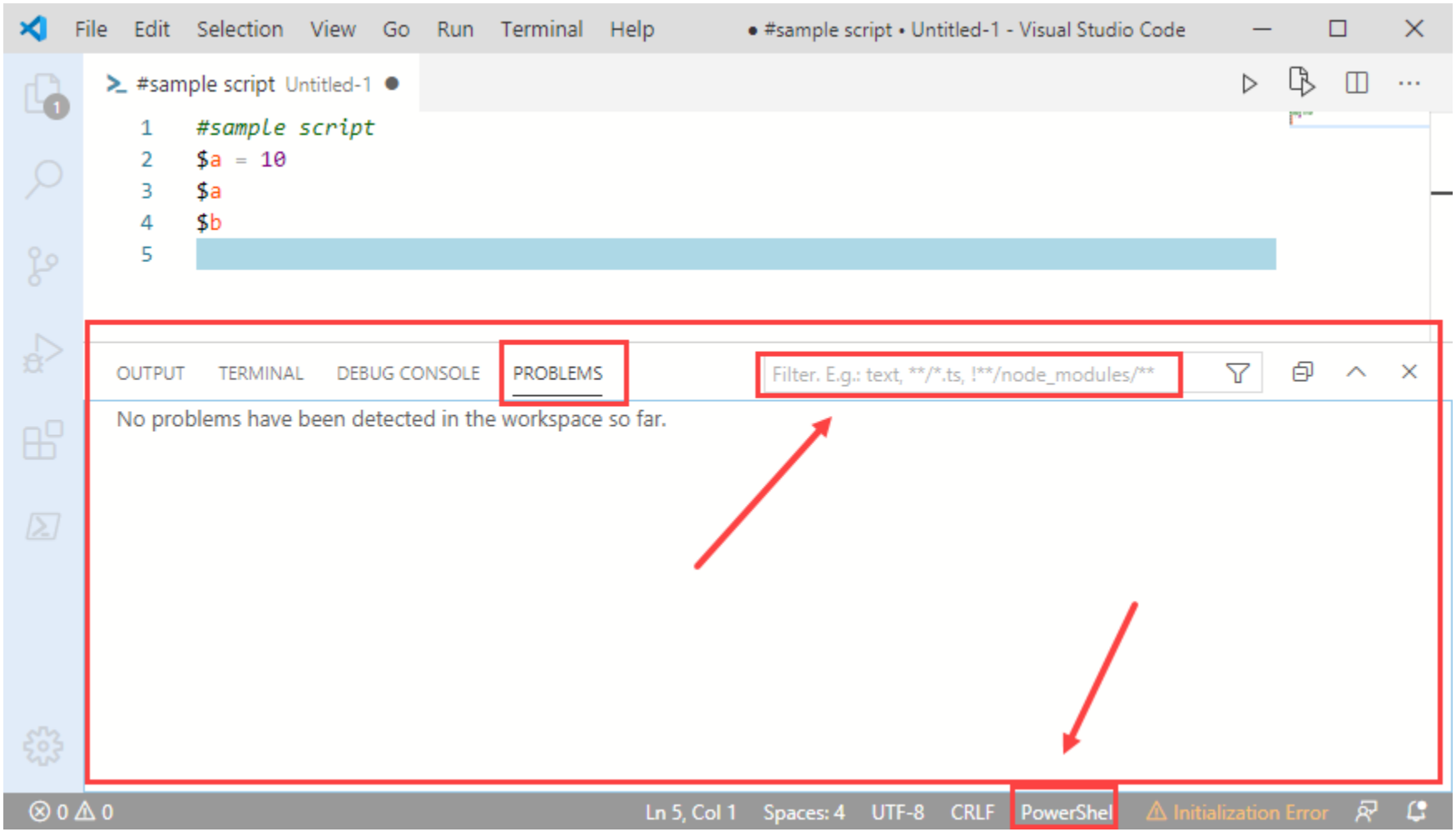
NOTE: the bottom window is loosely tied* to the top window (explained later)

Be mindful of the scope of the 'debug console' sub-window, as well as the language:



NOTE: the bottom window is loosely tied* to the top window (explained later)

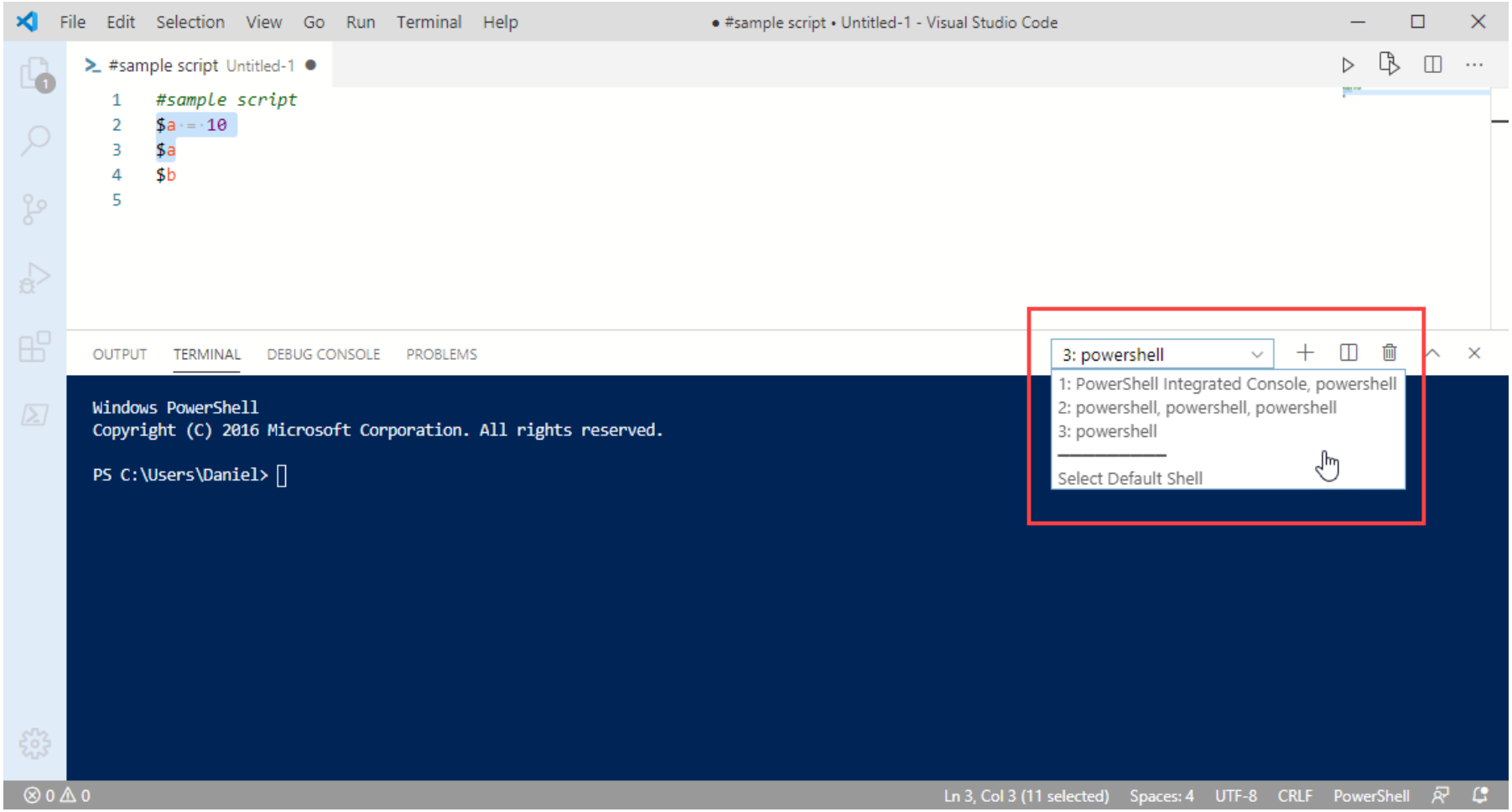
Be mindful of the scope of the ‘problems’ sub-window, as well as any filter, and the language:

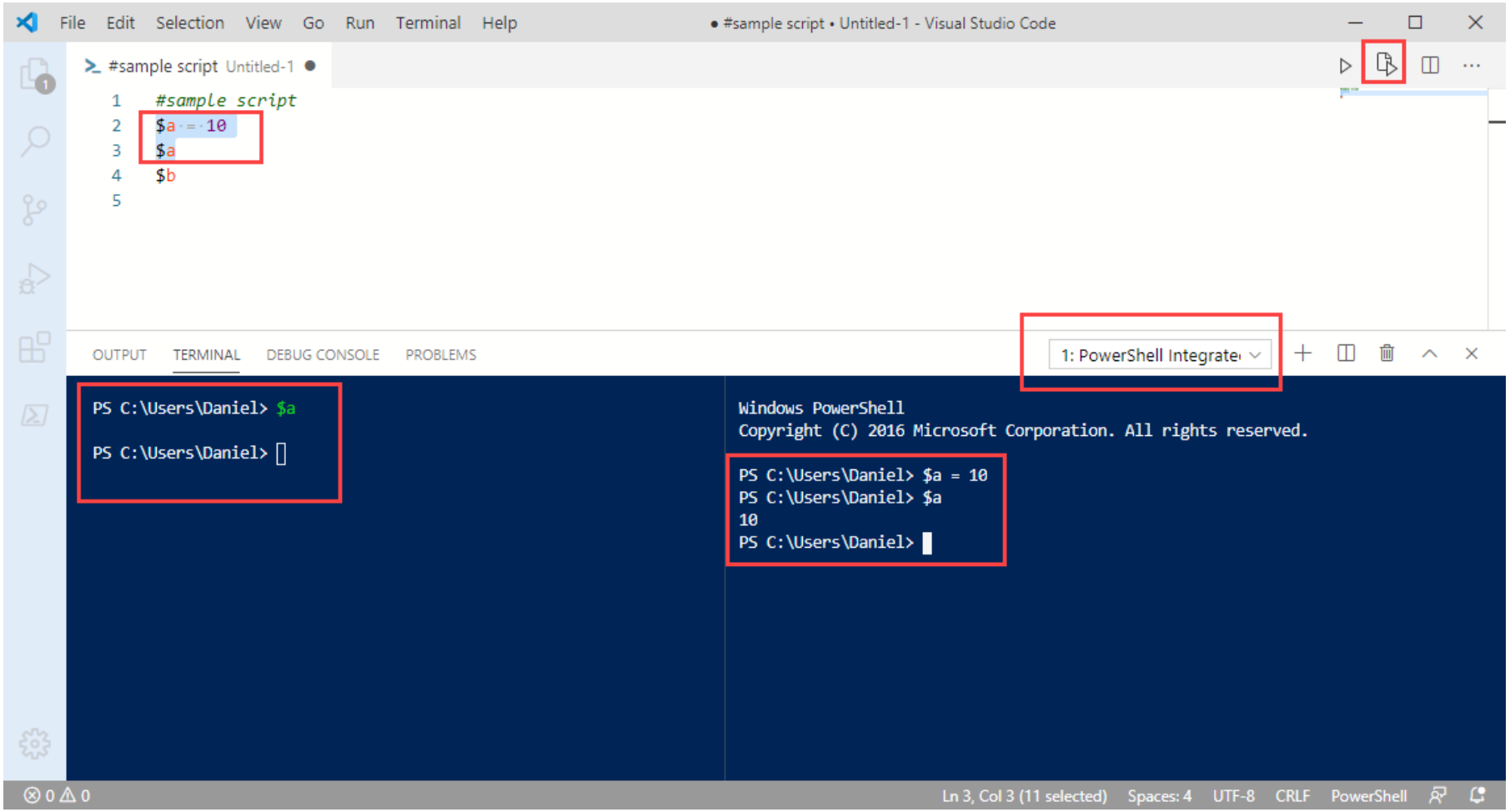


NOTE: the bottom window is loosely tied* to the top window (explained later)

Bottom Windows is Loosely Tied to Top Window:

- Each terminal is independent of any others, just like each PSh or CMD window on your workstation's desktop is independent of any others.
- Each terminal appears as a separate X: line in the scope drop-down.
  - All terminals can be split multiple times, which are also independent of one another.
  - All splits appear in the same X: line in the scope drop-down as their originating terminal.
- The 'PowerShell Integrated' terminal (and any of its splits) is loosely tied to the top/file window.
  - When the 'PowerShell Integrated' terminal is split, its splits are labeled as just 'powershell' but they are actually all 'PowerShell Integrated'.
  - When the 'PowerShell Integrated' terminal is split, only the terminal with focus will be affected by the top/file window.
- LC the trashcan icon to the upper right of the Terminal widow to kill the terminal with focus, and all of its data/variables (very handy).

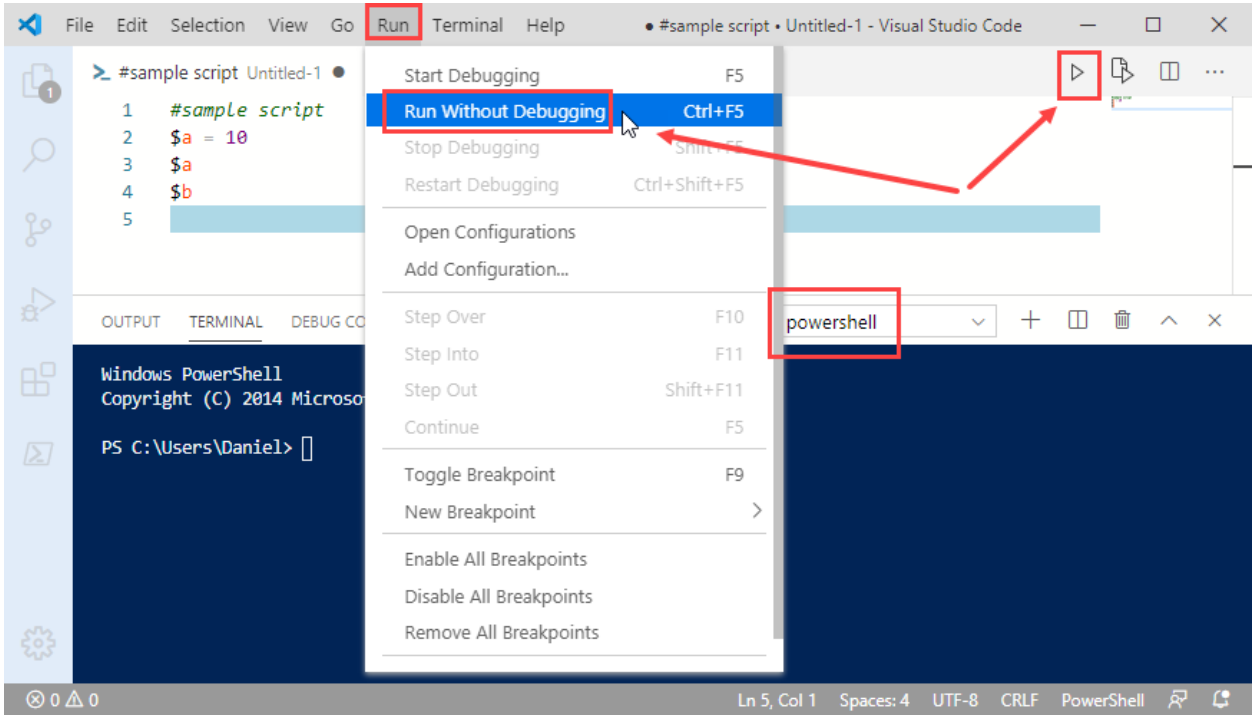



File Edit Selection View Go Run Terminal Help
• #sample script • Untitled-1 - Visual Studio Code
#sample script Untitled-1
1 #sample script
2 $a = 10
3 $a
4 $b
5
OUTPUT TERMINAL DEBUG CONSOLE PROBLEMS
1: PowerShell Integrate
PS C:\Users\Daniel> $a
PS C:\Users\Daniel>
Windows PowerShell
Copyright (C) 2016 Microsoft Corporation. All rights reserved.
PS C:\Users\Daniel> $a = 10
PS C:\Users\Daniel> $a
10
PS C:\Users\Daniel>
Ln 3, Col 3 (11 selected) Spaces: 4 UTF-8 CRLF PowerShell

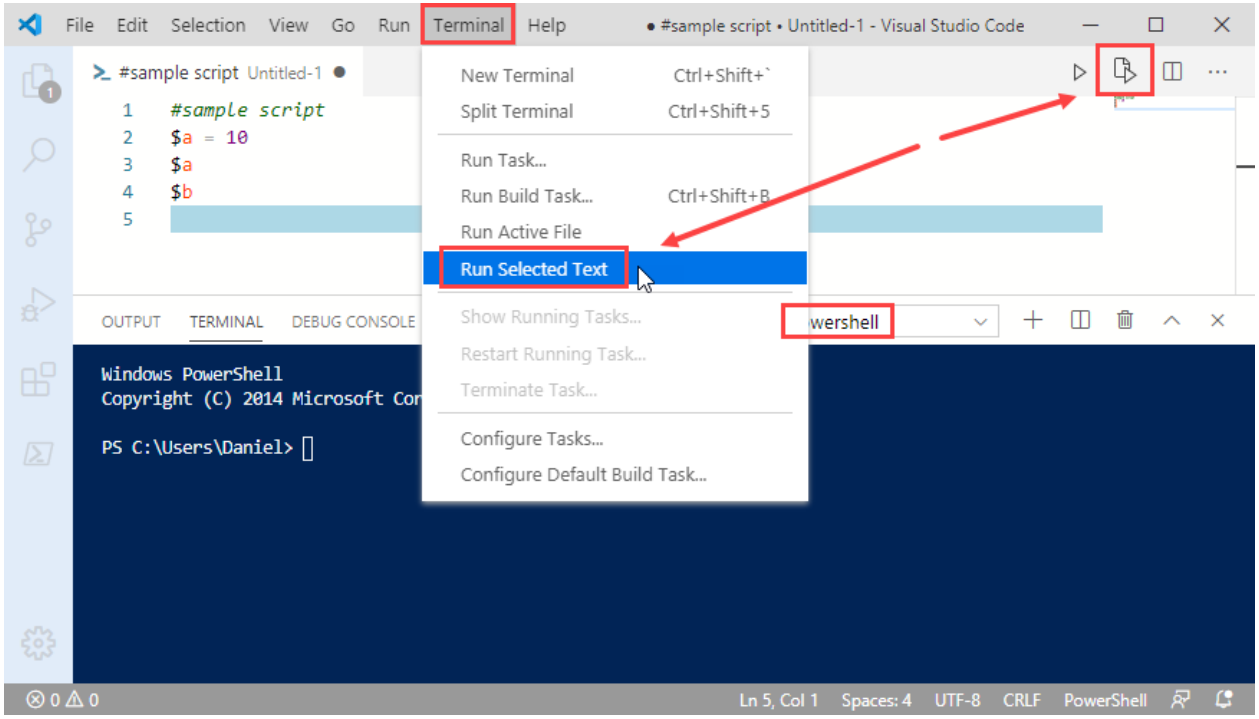
Debugging PSh Code with VSC:

(jump to TOC)

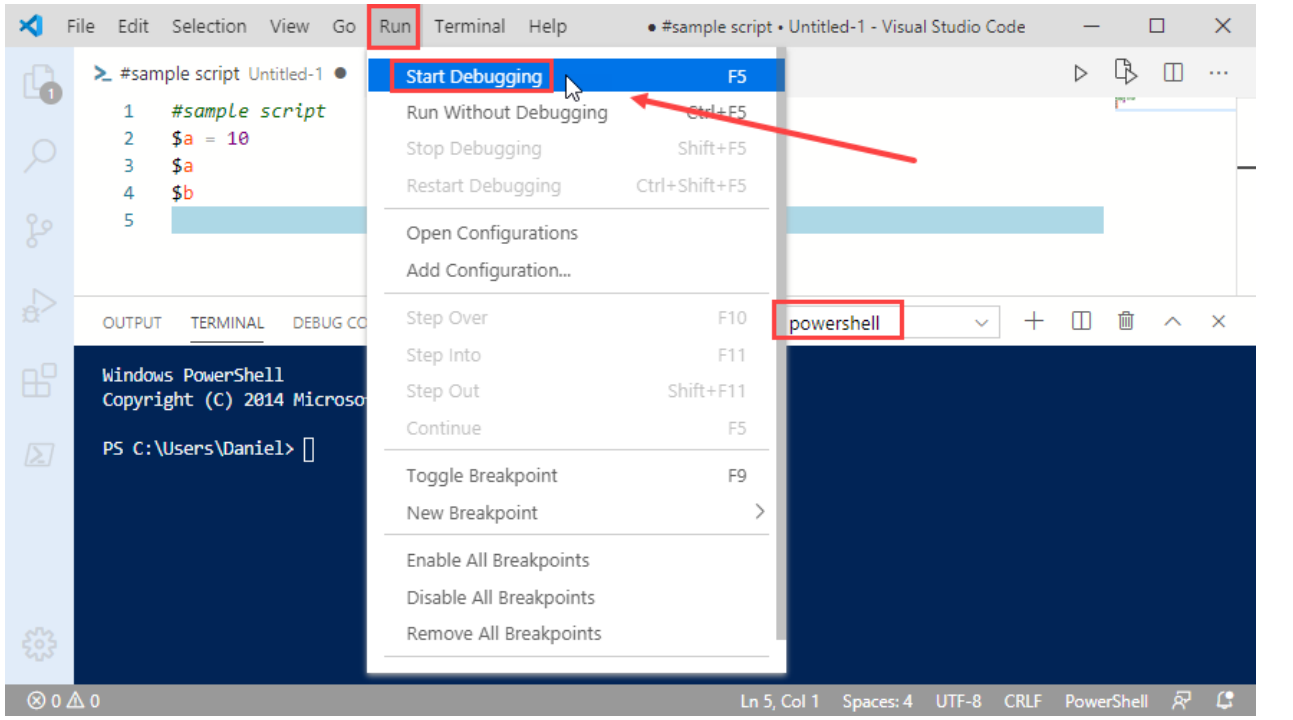
- To run all code, use the 'Run' PDM:



- To run selected code, use the'Terminal' PDM:



- Debugging:



move step over step out stop
continue step into restart

Like ISE:

- F5 run the script and start a debugging session
- F8 - run selection (LD to select, RC for context menu, LC run selection)
- F9 - toggles a breakpoint on a line of code
- F10 - step over
- F11 - step into
- Shft F11 - step out

NOTE: LC the trashcan icon to the upper right of the Terminal widow to kill that Terminal, and all data/variables in that Terminal session (very handy)

- continue will cause the debugger to **resume normal execution until the next breakpoint**, or the end of the script
- Step **over** **proceeds to the next line in your current scope (i.e. it goes to the next line), without descending into any method calls on the way**. This is generally used for following the logic through a particular method without worrying about the details of its collaborators, and can be useful for finding at what point in a method the expected conditions are violated.
- Step **into** will cause the debugger to **descend into any method calls on the current line**. If there are multiple method calls, they'll be visited in order of execution; if there are no method calls, this is same as step over. This is broadly equivalent to following every individual line of execution as would be seen by the interpreter.

- Step **out** **proceeds until the next "return" or equivalent** - i.e. until control has returned to the preceding stack frame. This is generally used when you've seen all you need to at *this* point/method, and want to bubble up the stack a few layers to where the value is actually used.
- restart - exact behavior has not been empirically tested by me
- stop - stop execution immediately

From <https://stackoverflow.com/questions/5391684/what-is-step-into-step-out-and-step-over-in-firebug>

Tutorials:

(jump to TOC)

- Help PDM > Introductory Videos - https://code.visualstudio.com/docs/getstarted/introvideos#VSCode)
  - https://www.youtube.com/watch?v=zhjU24hbYuI
    Become a PowerShell Debugging Ninja by Kirk Munro
  - https://www.youtube.com/watch?v=7ab4z9u7Q_I
    Debugging with Breakpoints in Visual Studio