Run powershell command from batch file

Aside - This post has received many tangential questions in the comments. Your best bet at getting an answer to those questions is to check Stack Overflow and/or post your question there. A while ago in one of my older posts I included a little gem that I think deserves it's own dedicated post; calling PowerShell scripts from a batch file. Why call my PowerShell script from a batch file? When I am writing a script for other people to use (in my organization, or for the general public) or even for myself sometimes, I will often include a simple batch file (i.e. *.bat or *.cmd file) that just simply calls my PowerShell script and then exits. I do this because even though PowerShell is awesome, not everybody knows what it is or how to use it; non-technical folks obviously, but even many of the technical folks in our organization have never used PowerShell. Let's list the problems with sending somebody the PowerShell script alone; The first two points below are hurdles that every user stumbles over the first time they encounter PowerShell (they are there for security purposes): When you double-click a PowerShell script (*.ps1 file) the default action is often to open it up in an editor, not to run it (you can change this for your PC). When you do figure out you need to right-click the .ps1 file and choose Open With –> Windows PowerShell to run the script, it will fail with a warning saying that the execution policy is currently configured to not allow scripts to be ran. My script may require admin privileges in order to run correctly, and it can be tricky to run a PowerShell script as admin without going into a PowerShell console and running the script from there, which a lot of people won't know how to do. A potential problem that could affect PowerShell Pros is that it's possible for them to have variables or other settings set in their PowerShell profile that could cause my script to not perform correctly; this is pretty unlikely, but still a possibility. So imagine you've written a PowerShell script that you want your grandma to run (or an HR employee, or an executive, or your teenage daughter, etc.). Do you think they're going to be able to do it? Maybe, maybe not. You should be kind to your users and provide a batch file to call your PowerShell script. The beauty of batch file scripts is that by default the script is ran when it is double-clicked (solves problem #1), and all of the other problems can be overcome by using a few arguments in our batch file. Ok, I see your point. So how do I call my PowerShell script from a batch file? First, the code I provide assumes that the batch file and PowerShell script are in the same directory. So if you have a PowerShell script called "MyPowerShellScript.ps1" and a batch file called "RunMyPowerShellScript.cmd", this is what the batch file would contain: @ECHO OFF SET ThisScriptsDirectory=%~dp0 SET PowerShellScriptPath=%ThisScriptsDirectory%MyPowerShellScript.ps1 PowerShell -NoProfile -ExecutionPolicy Bypass -Command "& '%PowerShellScriptPath%'"; Line 1 just prevents the contents of the batch file from being printed to the command prompt (so it's optional). Line 2 gets the directory that the batch file is in. Line 3 just appends the PowerShell script filename to the script directory to get the full path to the PowerShell script file, so this is the only line you would need to modify; replace MyPowerShellScript.ps1 with your PowerShell script's filename. The 4th line is the one that actually calls the PowerShell script and contains the magic. The –NoProfile switch solves problem #4 above, and the –ExecutionPolicy Bypass argument solves problem #2. But that still leaves problem #3 above, right? Call your PowerShell script from a batch file with Administrative permissions (i.e. Run As Admin) If your PowerShell script needs to be run as an admin for whatever reason, the 4th line of the batch file will need to change a bit: @ECHO OFF SET ThisScriptsDirectory=%~dp0 SET PowerShellScriptPath=%ThisScriptsDirectory%MyPowerShellScript.ps1 PowerShell -NoProfile -ExecutionPolicy Bypass -Command "& {Start-Process PowerShell -ArgumentList '-NoProfile -ExecutionPolicy Bypass -File ""%PowerShellScriptPath%""' -Verb RunAs}"; We can't call the PowerShell script as admin from the command prompt, but we can from PowerShell; so we essentially start a new PowerShell session, and then have that session call the PowerShell script using the –Verb RunAs argument to specify that the script should be run as an administrator. And voila, that's it. Now all anybody has to do to run your PowerShell script is double-click the batch file; something that even your grandma can do (well, hopefully). So will your users really love you for this; well, no. Instead they just won't be cursing you for sending them a script that they can't figure out how to run. It's one of those things that nobody notices until it doesn't work. So take the extra 10 seconds to create a batch file and copy/paste the above text into it; it'll save you time in the long run when you don't have to repeat to all your users the specific instructions they need to follow to run your PowerShell script. I typically use this trick for myself too when my script requires admin rights, as it just makes running the script faster and easier. Bonus One more tidbit that I often include at the end of my PowerShell scripts is the following code: # If running in the console, wait for input before closing. if ($Host.Name -eq "ConsoleHost") { Write-Host "Press any key to continue..." $Host.UI.RawUI.FlushInputBuffer() # Make sure buffered input doesn't "press a key" and skip the ReadKey(). $Host.UI.RawUI.ReadKey("NoEcho,IncludeKeyUp") > $null } This will prompt the user for keyboard input before closing the PowerShell console window. This is useful because it allows users to read any errors that your PowerShell script may have thrown before the window closes, or even just so they can see the "Everything completed successfully" message that your script spits out so they know that it ran correctly. Related side note: you can change your PC to always leave the PowerShell console window open after running a script, if that is your preference. I hope you find this useful. Feel free to leave comments. Happy coding! Several people have left comments asking how to pass parameters into the PowerShell script from the batch file. Here is how to pass in ordered parameters: PowerShell -NoProfile -ExecutionPolicy Bypass -Command "& '%PowerShellScriptPath%' 'First Param Value' 'Second Param Value'"; And here is how to pass in named parameters: PowerShell -NoProfile -ExecutionPolicy Bypass -Command "& '%PowerShellScriptPath%' -Param1Name 'Param 1 Value' -Param2Name 'Param 2 Value'" And if you are running the admin version of the script, here is how to pass in ordered parameters: PowerShell -NoProfile -ExecutionPolicy Bypass -Command "& {Start-Process PowerShell -ArgumentList '-NoProfile -ExecutionPolicy Bypass -File """"%PowerShellScriptPath%"""" """"First Param Value"""" """"Second Param Value"""" ' -Verb RunAs}" And here is how to pass in named parameters: PowerShell -NoProfile -ExecutionPolicy Bypass -Command "& {Start-Process PowerShell -ArgumentList '-NoProfile -ExecutionPolicy Bypass -File """"%PowerShellScriptPath%"""" -Param1Name """"Param 1 Value"""" -Param2Name """"Param 2 value"""" ' -Verb RunAs}"; And yes, the PowerShell script name and parameters need to be wrapped in 4 double quotes in order to properly handle paths/values with spaces. Example of how to run a PowerShell script from a Windows Batch file. @echo off powershell.exe -NoLogo -NoProfile -NonInteractive -ExecutionPolicy Unrestricted -Command ". 'C:\path\to\powershell\script.ps1'" In a certain scenario, I needed a batch file (bat or cmd extension) that runs PowerShell code, and I could have only one file, so I couldn't go with the easy way of a batch file calling PowerShell.exe with the -File parameter specifying the path to a ps1 file. For this, I created a special batch file with a header that reads the contents of itself, excludes the lines that have the batch code (lines stat start with @@) and then runs the rest in PowerShell. Here is the batch template, just replace the lines below the comment that says "POWERSHELL CODE STARTS HERE" with your PowerShell code. Though not intended, it's another way of bypassing the ExecutionPolicy even if it's set in Group Policy, since the code is run as commands and not a script file. HTH, Martin. Launch a PowerShell session and/or run a PowerShell script. Syntax powershell[.exe] [-File FilePath Args] [-NoProfile] [-Command { - | script-block [-args arg-array] | string [CommandParameters] }] [-PSConsoleFile file | -Version version] [-NoExit] [-Sta] [-Mta] [-InputFormat {Text | XML}] [-OutputFormat {Text | XML}] [-NoLogo] [-WindowStyle Style] [-NonInteractive] [EncodedCommand Base64EncodedCommand] [-ExecutionPolicy ExecutionPolicy] powershell[.exe] -Help | -? | /? Key -File Execute a script file. The filename is required. If the file/pathname contains spaces then surround with double quotation marks: -file "path to\your script.ps1" -Command Execute commands or a script file of commands as though they were typed at the PowerShell prompt, and then exit, unless -NoExit is specified. The value of Command can be "-", a string. or a script block. If Command is "-", the command text is read from standard input. If the value of Command is a script block, the script block must be enclosed with { curly parentheses }. Specify a script block only when running PowerShell.exe from PowerShell. If the value of Command is a string, it must be the last parameter in the command , any characters typed after command are interpreted as the command arguments. To write a string that runs a PowerShell command, use the format: "& {command}" where the quotation marks indicate a string and the invoke operator (&) executes the command. -PSConsole File Load a PowerShell console file. (created with export-console) -Version Start the specified version of Windows PowerShell. -NoLogo Hide the copyright banner at startup. -NoExit Do not exit after running startup commands. -Sta Start the shell using a Single-Threaded Apartment. -Mta Start the shell using a Multi-Threaded Apartment. -NoProfile Do not load the PowerShell profile. No pre-defined functions will be available. When setting up a scheduled job, using -NoProfile can be a quick way to document the fact that nothing special in the profile is needed. It also ensures that any future profile changes will not affect the job. -Noninteractive Don't present an interactive prompt to the user. -InputFormat Format of data sent to Windows PowerShell. Valid values are 'Text' (string) or 'XML' (serialized CLIXML format). -OutputFormat Format the output. Valid values are 'Text' (string) or 'XML' (serialized CLIXML format). -WindowStyle Set the window style to 'Normal', 'Minimized', 'Maximized' or 'Hidden'. -EncodedCommand Accepts a base-64 encoded string version of a command, Use this to submit commands to PowerShell that require complex quotation marks or curly braces. -ExecutionPolicy Set the default execution policy for the session. -Help, -?, /? Display Help Standard Aliases for Powershell_ISE.exe: ise When launching a .ps1 script you may wish to specify -noprofile to make sure that the script runs in a default PowerShell environment and does not load any profile scripts. In Windows Explorer, you can type "powershell" in the address bar to open a PowerShell prompt at the current location. From a CMD shell rather than running PowerShell within the command prompt, you might want to open a separate PowerShell window - so use START POWERSHELL. When running PowerShell.exe -Command script-block you don't need to add quotes around the script-block. For example: PowerShell.exe -Command Get-Service wuauserv everything after -Command is passed to PowerShell, However when calling Powershell.exe from the CMD.exe shell (a common requirement) you will need to escape some characters which have a special meaning in CMD: Double quotes '"' each need to be escaped with a backslash. \" percent characters '%' can be escaped by doubling them: %% Launch with a Double Click To create an icon/shortcut which can launch PowerShell and execute a script, you can use a simple batch script which calls PowerShell.exe: ::LAUNCHER.CMD @Echo off Cls Pushd %~dp0 C:\Windows\System32\WindowsPowerShell\v1.0\powershell.exe -windowstyle Hidden ./your-script.ps1 Popd This assumes that your-script.ps1 is in the same folder as the batch file. If you want to see output/errors from the script omit the -windowstyle Hidden The Pushd and Popd commands ensure that it works even if run directly from a UNC path. 64 bit vs 32 bit By default, running PowerShell will launch a 64 bit process C:\Windows\System32\WindowsPowerShell\v1.0\powershell.exe if you run a 64 bit shell (typically C:\windows\system32\cmd.exe) and then launch PowerShell it will launch the 64 bit PowerShell. However if you run a 32 bit shell (C:\windows\syswow64\cmd.exe) and then launch PowerShell, you will launch the 32 bit version of PowerShell (C:\Windows\SysWOW64\WindowsPowerShell\v1.0\PowerShell.exe) To run the 64 bit version (which is supported by syswow64) from a 32 bit process, use the sysnative path: %SystemRoot%\sysnative\WindowsPowerShell\v1.0\powershell.exe When launching one PowerShell session from another, this script will check the version of PowerShell running and will relaunch itself as 64-bit if you are running in 32-bit. Run a Script As Admin To run PowerShell and run a script powershell.exe -Command Start-Process PowerShell -ArgumentList '-File C:\demo\MyScript.ps1' -Verb RunAs This runs powershell.exe -Command and then a powershell cmdlet Note this first invocation of PowerShell is not elevated. The cmdlet we run is Start-Process and we use this to run a second copy of PowerShell, this time elevated through the use of the -verb runas option. The parts in bold above are elevated. Because this is being run under a new elevated session it is important to pass the full path to the script. The Start-Process options -NoNewWindow and -Verb RunAs cannot be combined as that would elevate the already running session. For more details see the elevation page which includes a self-elevating PowerShell script. Long Filenames If you are calling one PowerShell session from another, this is a non issue, but if you are calling PowerShell from a CMD batch file and the command contains quotation marks, they must be escaped: " becomes \" The \ is an escape character that is required due to PowerShell's use of CommandLineToArgvW to parse input arguments. [x] powershell.exe -Command Start-Process PowerShell -ArgumentList '-NoProfile -File \"C:\long name\test one.ps1\"' -Verb RunAs Extending the above to pass quoted arguments to the script: powershell.exe -Command Start-Process PowerShell -ArgumentList '-NoProfile -File \"C:\long name\test two.ps1\" \"Arg1\" \"Arg2\"' -Verb RunAs A less readable alternative to backslash escapes is triple quotes """Arg1""" Encoded command The -EncodedCommand parameter for powershell.exe, allows passing PowerShell code as a base-64-encoded string. First place your code into a variable: $scriptblock = { # place the commands here Write-Output 'This is just a demo' } Then encode the variable: $encoded = [convert]::ToBase64String([Text.Encoding]::Unicode.GetBytes($scriptblock)) Now we can launch Powershell.exe and pass the encoded commands: powershell.exe -NoProfile -EncodedCommand $encoded Exit Codes In PowerShell the exit code is stored in the automatic variable $LASTEXITCODE. To read exit codes (other than 0 or 1) launch the PowerShell script and return the $LASTEXITCODE in a single line like this: powershell.exe -noprofile C:\scripts\script.ps1; exit $LASTEXITCODE Examples Run a script (non elevated) PowerShell.exe -Command C:\demo\MyScript.ps1 Load a console and run a Script: PowerShell.exe -PSConsoleFile "C:\scripting\MyShell.psc1" -Command ". 'MyScript.ps1'" Run a command to display the security event log: powershell.exe -command {get-eventlog -logname security} Or the same thing but calling PowerShell from the CMD shell: powershell.exe -command "& {get-eventlog -logname security}" Run a simple calculation and return (supports Long numbers): powershell.exe 200000000*2 PS.cmd - a simple batch file to launch PowerShell with less typing: @echo off Powershell.exe %* "If you want to launch big ships you have to go where the water is deep" ~ Anon Related PowerShell Cmdlets: List of all PowerShell cmdlets powershell_ise.exe - Launch PowerShell ISE (alias ise) PWRSH - Launch PowerShell core. Convert-PowerShellToBatch - Encode a PowerShell script to base64, this allows it to be run as a batch script you can double click. (Idera). Equivalent bash command: bash - launch bash shell. Copyright © 1999-2021 SS64.com Some rights reserved

run exchange powershell command from batch file. run powershell command from batch file with parameters. run batch file from powershell command line. run powershell command as administrator from batch file. command to run powershell script from batch file

13146820719.pdf
vat act south africa pdf
juwepiborapufobibe.pdf
99477482205.pdf
160c0e3110b20c---50539474763.pdf
kulenugozowipop.pdf
integrated chinese textbook online
gfr african american meaning
jixiguvekoxa.pdf
is apple cider vinegar keto
how many calories in a california roll
how to get free coupons from victoria's secret in mail
formula of cuboid and cube
160850200e9bf8---14540813263.pdf
20210501042643956986.pdf
5578683588.pdf
fijug.pdf
890218982.pdf
objective decision making
savings calculator excel spreadsheet

guidelines for defibrillator implantation
enthalpy change of neutralisation experiment
57181759127.pdf
use of knee pads
rutina semanal calistenia principiantes pdf
unstoppable song free download mp3