about_PowerShell_exe - PowerShell

By sdwheeler

Archived: 2026-04-05 19:28:01 UTC

SHORT DESCRIPTION

Explains how to use the `powershell.exe` command-line interface. Displays the command-line parameters and describes the syntax.

LONG DESCRIPTION

For information about the command-line options for PowerShell 7, see about_Pwsh.

Syntax

```
PowerShell[.exe]
   [-PSConsoleFile <file> | -Version <version>]
   [-NoLogo]
   [-NoExit]
   [-Sta]
   [-Mta]
   [-NoProfile]
   [-NonInteractive]
   [-InputFormat {Text | XML}]
   [-OutputFormat {Text | XML}]
   [-WindowStyle <style>]
   [-EncodedArguments <Base64EncodedArguments>]
   [-EncodedCommand <Base64EncodedCommand>]
   [-ConfigurationName <string>]
   [-File - | <filePath> <args>]
   [-ExecutionPolicy <ExecutionPolicy>]
   [-Command - | { <script-block> [-args <arg-array>] }
               | { <string> [<CommandParameters>] } ]

PowerShell[.exe] -Help | -? | /?
```

Parameters

All parameters are case-insensitive.

-File - | <filePath> <args>

The value of **File** can be `-` or a filepath and optional parameters. If the value of **File** is `-`, then commands are read from standard input.

If the value of **File** is a filepath, the script runs in the local scope ("dot-sourced") of the new session, so that the functions and variables that the script creates are available in that new session. Enter the script filepath and any parameters. **File** *must* be the last parameter in the command. All values typed after the **File** parameter are interpreted as the script filepath and parameters passed to that script. For example: `-File .\Get-Script.ps1 -Domain Central`

Typically, the switch parameters of a script are either included or omitted. For example, the following command uses the **All** parameter of the `Get-Script.ps1` script file: `-File .\Get-Script.ps1 -All`

In rare cases, you might need to provide a **Boolean** value for a parameter. It isn't possible to pass an explicit boolean value for a switch parameter when running a script in this way. This limitation was removed in PowerShell 6 (`pwsh.exe`).

Parameters passed to the script are passed as literal strings, after interpretation by the current shell. For example, if you are in `cmd.exe` and want to pass an environment variable value, you would use the `cmd.exe` syntax: `powershell.exe -File .\test.ps1 -TestParam %windir%`

In contrast, running `powershell.exe -File .\test.ps1 -TestParam $Env:windir` in `cmd.exe` results in the script receiving the literal string `$Env:windir` because it has no special meaning to the current `cmd.exe` shell. The `$Env:windir` style of environment variable reference *can* be used inside a **Command** parameter, since there it's interpreted as PowerShell code.

Similarly, if you want to execute the same command from a *Batch script*, you would use `%~dp0` instead of `.\` or `$PSScriptRoot` to represent the current execution directory: `pwsh -File %~dp0test.ps1 -TestParam %windir%`. If you use `.\test.ps1` instead, PowerShell throws an error because it can't find the literal path `.\test.ps1`

Note

The **File** parameter can't support scripts using a parameter that expects an array of argument values. This, unfortunately, is a limitation of how a native command gets argument values. When you call a native executable (such as `powershell` or `pwsh`), it doesn't know what to do with an array, so it's passed as a string.

If the value of **File** is `-`, then commands are read from standard input. Running `powershell -File -` without redirected standard input starts a regular session. This is the same as not specifying the `File` parameter at all. When reading from standard input, the input statements are executed one statement at a time as though they were typed at the PowerShell command prompt. If a statement doesn't parse correctly, the statement isn't executed. The process exit code is determined by status of the last (executed) command. With successful execution, the exit code is always `0`. When the script file terminates with an `exit` command, the process exit code is set to the numeric argument used with the `exit` command.

Similar to `-Command`, when a script-terminating error occurs, the exit code is set to `1`. However, unlike with `-Command`, when the execution is interrupted with `Ctrl+C` the exit code is `0`. For more information, see

`$LASTEXITCODE` in about_Automatic_Variables.

-Command

The value of **Command** can be `-`, a script block, or a string. If the value of **Command** is `-`, the command text is read from standard input.

The **Command** parameter only accepts a script block for execution when it can recognize the value passed to **Command** as a **ScriptBlock** type. This is *only* possible when running `powershell.exe` from another PowerShell host. The **ScriptBlock** type may be contained in an existing variable, returned from an expression, or parsed by the PowerShell host as a literal script block enclosed in curly braces (`{}`), before being passed to `powershell.exe`.

```
powershell -Command {Get-WinEvent -LogName Security}
```

In `cmd.exe`, there is no such thing as a script block (or **ScriptBlock** type), so the value passed to **Command** is *always* a string. You can write a script block inside the string, but instead of being executed it behaves exactly as though you typed it at a typical PowerShell prompt, printing the contents of the script block back out to you.

A string passed to **Command** is still executed as PowerShell code, so the script block curly braces are often not required in the first place when running from `cmd.exe`. To execute an inline script block defined inside a string, the call operator `&` can be used:

```
powershell.exe -Command "& {Get-WinEvent -LogName Security}"
```

If the value of **Command** is a string, **Command** must be the last parameter for pwsh, because all arguments following it are interpreted as part of the command to execute.

When called from within an existing PowerShell session, the results are returned to the parent shell as deserialized XML objects, not live objects. For other shells, the results are returned as strings.

If the value of **Command** is `-`, the commands are read from standard input. You must redirect standard input when using the **Command** parameter with standard input. For example:

```
 @'
"in"

"hi" |
  % { "$_ there" }

"out"
'@ | powershell -NoProfile -Command -
```

This example produces the following output:

```
 in
hi there
out
```

When reading from standard input, the input is parsed and executed one statement at a time, as though they were typed at the PowerShell command prompt. If the input code doesn't parse correctly, the statement isn't executed. Unless you use the `-NoExit` parameter, the PowerShell session exits when there is no more input to read from standard input.

The process exit code is determined by status of the last (executed) command within the input. The exit code is `0` when `$?` is `$true` or `1` when `$?` is `$false`. If the last command is an external program or a PowerShell script that explicitly sets an exit code other than `0` or `1`, that exit code is converted to `1` for process exit code. Similarly, the value 1 is returned when a script-terminating (runspace-terminating) error, such as a `throw` or `-ErrorAction Stop`, occurs or when execution is interrupted with `Ctrl+C`.

To preserve the specific exit code, add `exit $LASTEXITCODE` to your command string or script block. For more information, see `$LASTEXITCODE` in about_Automatic_Variables.

-ConfigurationName <string>

Specifies a configuration endpoint in which PowerShell is run. This can be any endpoint registered on the local machine including the default PowerShell remoting endpoints or a custom endpoint having specific user role capabilities.

-EncodedArguments <Base64EncodedArguments>

Accepts a Base64-encoded string version command arguments. Use this parameter to submit arguments that require complex, nested quoting. The Base64 representation must be a UTF-16LE encoded string.

-EncodedCommand <Base64EncodedCommand>

Accepts a base-64-encoded string version of a command. Use this parameter to submit commands to PowerShell that require complex quotation marks or curly braces. The string must be formatted using UTF-16LE character encoding.

-ExecutionPolicy <ExecutionPolicy>

Sets the default execution policy for the current session and saves it in the `$Env:PSExecutionPolicyPreference` environment variable. This parameter does not change the PowerShell execution policy that's set in the registry. For information about PowerShell execution policies, including a list of valid values, see about_Execution_Policies.

-InputFormat {Text | XML}

Describes the format of data sent to PowerShell. Valid values are `Text` (text strings) or `XML` (serialized CLIXML format).

-Mta

Starts PowerShell using a multi-threaded apartment. This parameter is introduced in PowerShell 3.0. In PowerShell 2.0, multi-threaded apartment (MTA) is the default. In PowerShell 3.0, single-threaded apartment (STA) is the default.

-NoExit

Doesn't exit after running startup commands.

-NonInteractive

This switch is used to create sessions that shouldn't require user input. This is useful for scripts that run in scheduled tasks or CI/CD pipelines. Any attempts to use interactive features, like `Read-Host` or confirmation prompts, result in statement terminating errors rather than hanging.

-NoLogo

Hides the copyright banner at startup.

-NoProfile

Doesn't load the PowerShell profile.

-OutputFormat {Text | XML}

Determines how output from PowerShell is formatted. Valid values are `Text` (text strings) or `XML` (serialized CLIXML format).

-PSConsoleFile <FilePath>

Loads the specified PowerShell console file. Enter the path and name of the console file. To create a console file, use the Export-Console cmdlet in PowerShell.

-Sta

Starts PowerShell using a single-threaded apartment. In Windows PowerShell 2.0, multi-threaded apartment (MTA) is the default. In Windows PowerShell 3.0, single-threaded apartment (STA) is the default.

-Version <PowerShell Version>

Starts the specified version of PowerShell. Valid values are 2.0 and 3.0. The version that you specify must be installed on the system. If Windows PowerShell 3.0 is installed on the computer, "3.0" is the default version. Otherwise, "2.0" is the default version. For more information, see Installing PowerShell.

-WindowStyle <Window style>

Sets the window style for the session. Valid values are `Normal`, `Minimized`, `Maximized`, and `Hidden`.

-Help, -?, /?

Displays help for `powershell.exe`. If you are typing a `powershell.exe` command in a PowerShell session, prepend the command parameters with a hyphen (`-`), not a forward slash (`/`). You can use either a hyphen or forward slash in `cmd.exe`.

Troubleshooting note: In PowerShell 2.0, starting some programs from the PowerShell console fails with a **LastExitCode** of 0xc0000142.

Examples

```
 # Create a new PowerShell session and load a saved console file
PowerShell -PSConsoleFile sqlsnapin.psc1

# Create a new PowerShell V2 session with text input, XML output, and no logo
PowerShell -Version 2.0 -NoLogo -InputFormat text -OutputFormat XML

# Execute a PowerShell Command in a session
PowerShell -Command "Get-EventLog -LogName Security"

# Run a script block in a session
PowerShell -Command {Get-EventLog -LogName Security}

# An alternate way to run a command in a new session
PowerShell -Command "& {Get-EventLog -LogName Security}"

# To use the -EncodedCommand parameter:
$command = "dir 'C:\Program Files' "
$bytes = [System.Text.Encoding]::Unicode.GetBytes($command)
$encodedCommand = [Convert]::ToBase64String($bytes)
powershell.exe -EncodedCommand $encodedCommand
```
