PowerShell differences on non-Windows platforms - PowerShell

sdwheeler ⋮ 14-17 minutes

PowerShell differences on non-Windows platforms

- Article
- 05/13/2024
-

In this article

PowerShell strives to provide feature parity across all supported platforms. However, some features behave differently or aren't available due to differences in .NET Core and platform-specific differences. Other changes were made to improve the interoperability of PowerShell on non-Windows platforms.

.NET Framework vs .NET Core

PowerShell on Linux and macOS uses .NET Core, a subset of the full .NET Framework on Microsoft Windows. As a result, scripts that run on Windows might not run on non-Windows platforms because of the differences in the frameworks.

For more information about changes in .NET Core, see Breaking changes for migration from .NET Framework to .NET Core.

General Unix interoperability changes

- Added support for native command globbing on Unix platforms. This means you can use wildcards with native commands like `ls *.txt`.
- The `more` functionality respects the Linux `$PAGER` and defaults to `less`.
- Trailing backslash is automatically escaped when dealing with native command arguments.
- Fixed ConsoleHost to honor `NoEcho` on Unix platforms.
- Don't add `PATHEXT` environment variable on Unix.
- A `powershell` man-page is included in the package.

Execution policy

PowerShell ignores execution policies when running on non-Windows platforms. `Get-ExecutionPolicy` returns **Unrestricted** on Linux and macOS. `Set-ExecutionPolicy` does nothing on Linux and macOS.

Case-sensitivity in PowerShell

Historically, PowerShell has been uniformly case-insensitive, with few exceptions. On UNIX-like operating systems, the file system is predominantly case-sensitive, and PowerShell adheres to the standard of the file system.

- You must use the correct case when a filename in specified in PowerShell.
- If a script tries to load a module and the module name isn't cased correctly, then the module load fails. This behavior might cause a problem with existing scripts if the name referenced by the module doesn't match the proper case of the actual filename.
- While names in the filesystem are case-sensitive, tab-completion of filenames isn't case-sensitive. Tab-completion cycles through the list of names using case-insensitive matching.
- `Get-Help` supports case-insensitive pattern matching on Unix platforms.
- `Import-Module` is case insensitive when used with a filename to determine the module name.

Filesystem support for Linux and macOS

- Paths given to cmdlets are now slash-agnostic (both `/` and `\` work as directory separators)
- XDG Base Directory Specification is now respected and used by default:
  - The Linux/macOS profile path is located at `~/.config/powershell/profile.ps1`
  - The history save path is located at `~/.local/share/powershell/PSReadline/ConsoleHost_history.txt`
  - The user module path is located at `~/.local/share/powershell/Modules`
- Support for file and folder names containing the colon character on Unix.
- Support for script names or full paths that have commas.
- Detect when the **LiteralPath** parameter is used to suppress wildcard expansion for navigation cmdlets.
- Updated `Get-ChildItem` to work more like the *nix `ls -R` and the Windows `DIR /S` native commands. `Get-ChildItem` now returns the symbolic links encountered during a recursive search and doesn't search the directories that those links target.

.PS1 File Extensions

PowerShell scripts must end in `.ps1` for the interpreter to understand how to load and run them in the current process. Running scripts in the current process is the expected usual behavior for PowerShell. You can add the `#!` magic number to a script that doesn't have a `.ps1` extension, but this causes the script to be run in a new PowerShell instance, preventing the script from working correctly when interchanging objects. This behavior might be desirable when executing a PowerShell script from Bash or another shell.

Convenience aliases removed

PowerShell provides a set of aliases on Windows that map to Linux command names for user convenience. On Linux and macOS, the "convenience aliases" for the basic commands `ls`, `cp`, `mv`, `rm`, `cat`, `man`, `mount`, and `ps` were removed to allow the native executable to run without specifying a path.

Logging

On macOS, PowerShell uses the native `os_log` APIs to log to Apple's unified logging system. On Linux, PowerShell uses Syslog, a ubiquitous logging solution.

Job Control

There's no Unix-style job-control support in PowerShell on Linux or macOS. The `fg` and `bg` commands aren't available. However, you can use PowerShell jobs that work on all platforms.

Putting `&` at the end of a pipeline causes the pipeline to be run as a PowerShell job. When a pipeline is backgrounded, a job object is returned. Once the pipeline is running as a job, all `*-Job` cmdlets can be used to manage the job. Variables (ignoring process-specific variables) used in the pipeline are automatically copied to the job so `Copy-Item $foo $bar &` just works. The job runs in the current directory instead of the user's home directory.

Remoting Support

PowerShell Remoting (PSRP) using WinRM on Unix platforms requires NTLM/Negotiate or Basic Auth over HTTPS. PSRP on macOS only supports Basic Auth over HTTPS. Kerberos-based authentication isn't supported.

PowerShell supports PowerShell Remoting (PSRP) over SSH on all platforms (Windows, Linux, and macOS). For more information, see SSH remoting in PowerShell.

Just-Enough-Administration (JEA) Support

PowerShell on Linux or macOS doesn't allow you to create constrained administration (JEA) remoting endpoints.

`sudo`, `exec`, and PowerShell

Because PowerShell runs most commands in memory (like Python or Ruby), you can't use `sudo` directly with PowerShell built-ins. You can run `pwsh` from `sudo`. If it's necessary to run a PowerShell cmdlet from within PowerShell with `sudo`, for example, `sudo Set-Date 8/18/2016`, then you would use `sudo pwsh Set-Date 8/18/2016`.

Modules included on non-Windows platforms

For non-Windows platforms, PowerShell includes the following modules:

- Microsoft.PowerShell.Archive
- Microsoft.PowerShell.Core
- Microsoft.PowerShell.Host
- Microsoft.PowerShell.Management
- Microsoft.PowerShell.Security
- Microsoft.PowerShell.Utility
- PackageManagement
- PowerShellGet
- PSReadLine
- ThreadJob

A large number of the commands (cmdlets) commonly available in PowerShell aren't available on Linux or macOS. Often, these commands don't apply to these platforms. For example, commands for Windows-specific features like the registry or services aren't available. Other commands, like `Set-ExecutionPolicy`, are present but not functional.

For a comprehensive list of modules and cmdlets and the platforms they support, see Release history of modules and cmdlets.

Modules no longer shipped with PowerShell

For various compatibility reasons, the following modules are no longer included in PowerShell.

- ISE
- Microsoft.PowerShell.LocalAccounts
- Microsoft.PowerShell.ODataUtils
- Microsoft.PowerShell.Operation.Validation
- PSScheduledJob
- PSWorkflow
- PSWorkflowUtility

The following Windows-specific modules aren't included in PowerShell for Linux or macOS.

- CimCmdlets
- Microsoft.PowerShell.Diagnostics
- Microsoft.WSMan.Management
- PSDiagnostics

Cmdlets not available on non-Windows platforms

Some cmdlets were removed from PowerShell. Others aren't available or might work differently on non-Windows platforms. For a comprehensive list of cmdlets removed from PowerShell, see Cmdlets removed from PowerShell.

Microsoft.PowerShell.Core

The following cmdlets aren't available on Linux or macOS:

- `Disable-PSRemoting`
- `Enable-PSRemoting`
- `Connect-PSSession`
- `Disconnect-PSSession`
- `Receive-PSSession`
- `Get-PSSessionCapability`

- `Disable-PSSessionConfiguration`
- `Enable-PSSessionConfiguration`
- `Get-PSSessionConfiguration`
- `Register-PSSessionConfiguration`
- `Set-PSSessionConfiguration`
- `Unregister-PSSessionConfiguration`
- `Test-PSSessionConfigurationFile`

The **ShowWindow** parameter of `Get-Help` isn't available for non-Windows platforms. PowerShell 7.3 added the `Switch-Process` cmdlet and the `exec` function for Linux and macOS. These commands aren't available on Windows.

Microsoft.PowerShell.Security cmdlets

The following cmdlets aren't available on Linux or macOS:

- `Get-Acl`
- `Set-Acl`
- `Get-AuthenticodeSignature`
- `Set-AuthenticodeSignature`
- `New-FileCatalog`
- `Test-FileCatalog`

These cmdlets are only available beginning in PowerShell 7.1.

- `Get-CmsMessage`
- `Protect-CmsMessage`
- `Unprotect-CmsMessage`

Microsoft.PowerShell.Management cmdlets

The following cmdlets aren't available on Linux and macOS:

- `Rename-Computer`
- `Get-ComputerInfo`
- `Get-HotFix`
- `Clear-RecycleBin`
- `Get-Service`
- `New-Service`
- `Remove-Service`
- `Restart-Service`
- `Resume-Service`
- `Set-Service`
- `Start-Service`
- `Stop-Service`
- `Suspend-Service`
- `Set-TimeZone`

The following cmdlets are available with limitations:

- `Get-Clipboard` - available in PowerShell 7.0+
- `Set-Clipboard` - available in PowerShell 7.0+

- `Restart-Computer` - available for Linux and macOS in PowerShell 7.1+
- `Stop-Computer` - available for Linux and macOS in PowerShell 7.1+

Microsoft.PowerShell.Utility cmdlets

The following cmdlets aren't available on Linux and macOS:

- `Convert-String`
- `ConvertFrom-String`
- `ConvertFrom-SddlString`
- `Out-GridView`
- `Out-Printer`
- `Show-Command`

Aliases not available on Linux or macOS

The following table lists the aliases available for Windows that aren't available on non-Windows platforms. These aliases aren't available because the alias conflicts with a native command on those platforms.

Alias	Cmdlet
`ac`	`Add-Content`
`cat`	`Get-Content`
`clear`	`Clear-Host`
`compare`	`Compare-Object`
`cp`	`Copy-Item`
`cpp`	`Copy-ItemProperty`
`diff`	`Compare-Object`
`kill`	`Stop-Process`
`ls`	`Get-ChildItem`
`man`	`help`
`mount`	`New-PSDrive`
`mv`	`Move-Item`
`ps`	`Get-Process`
`rm`	`Remove-Item`
`rmdir`	`Remove-Item`
`sleep`	`Start-Sleep`

Alias	Cmdlet
`sort`	`Sort-Object`
`start`	`Start-Process`
`tee`	`Tee-Object`
`write`	`Write-Output`

The table doesn't include aliases unavailable for cmdlets that don't exist on non-Windows platforms.

PowerShell Desired State Configuration (DSC)

Beginning with PowerShell 7.2, the **PSDesiredStateConfiguration** module was removed from PowerShell and is published in the PowerShell Gallery. For more information, see the announcement on the PowerShell Team blog. For more information about using DSC on Linux, see Get started with DSC for Linux. DSC v1.1 and v2.x aren't supported on macOS. DSC v3 is supported on Windows, Linux, and macOS, but it's still in early development.

Additional resources

Training

Documentation

- Release history of modules and cmdlets - PowerShell

  This article lists the modules and cmdlets that are included in various versions of PowerShell.

- about_Windows_PowerShell_Compatibility - PowerShell

  Describes the Windows PowerShell Compatibility functionality for PowerShell 7.

- about_PowerShell_Editions - PowerShell

  Different editions of PowerShell run on different underlying runtimes.

- PowerShell Support Lifecycle - PowerShell

  Details the policies governing support for PowerShell.

- What's New in PowerShell 7.4 - PowerShell

  New features and changes released in PowerShell 7.4

In this article

Previous Chapter Next Chapter