

PowerShell

PowerShell is a task automation and configuration management framework from [Microsoft](#), consisting of a [command-line shell](#) and associated [scripting language](#). Initially a Windows component only, known as **Windows PowerShell**, it was made [open-source](#) and [cross-platform](#) on 18 August 2016 with the introduction of **PowerShell Core**.^[5] The former is built on [.NET Framework](#) while the latter on [.NET Core](#).

In PowerShell, administrative tasks are generally performed by *cmdlets* (pronounced *command-lets*), which are specialized [.NET classes](#) implementing a particular operation. These work by accessing data in different data stores, like the [file system](#) or [registry](#), which are made available to PowerShell via *providers*. Third-party developers can add cmdlets and providers to PowerShell.^{[6][7]} Cmdlets may be used by *scripts* and scripts may be packaged into *modules*.

PowerShell provides full access to [COM](#) and [WMI](#), enabling administrators to perform administrative tasks on both local and remote Windows systems as well as [WS-Management](#) and [CIM](#) enabling management of remote Linux systems and network devices. PowerShell also provides a hosting [API](#) with which the PowerShell runtime can be embedded inside other applications. These applications can then use PowerShell functionality to implement certain operations, including those exposed via the [graphical interface](#). This capability has been used by [Microsoft Exchange Server 2007](#) to expose its management functionality as PowerShell cmdlets and providers and implement the [graphical management tools](#) as PowerShell hosts which invoke the necessary cmdlets.^{[6][8]} Other Microsoft applications including [Microsoft SQL Server 2008](#) also expose their management interface via PowerShell cmdlets.^[9]

PowerShell includes its own extensive, console-based help (similar to [man pages](#) in [Unix shells](#)) accessible via the `Get -Help` cmdlet. Local help contents can be retrieved from the Internet via `Update-Help` cmdlet. Alternatively, help from the web can be acquired on a case-by-case basis via the `-online` switch to `Get -Help`.

Contents

Background

PowerShell



```
PS C:\> Get-Childitem 'MediaCenter\Music' -rec |
>> where { -not $_.FileContainer -and $_.Extension -match '.m3p3' } |
>> Measure-Object -property length -sum -min -max -ave

Count      : 1387
Average     : 5491276.89563887
Sum         : 71778977857
Maximum     : 227865267
Minimum     : 3235
Property    : Length

PS C:\> Get-WmiObject CIM_BIOElement | select bios*, name, size | Format-List

BIOVersion   : (TOECPL - 6040000, Ver 1.0MPARTBL)
Manufacturer : TOSHIBA
SerialNumber  : ME211168

PS C:\> (wmiSearcher)'
>> SELECT * FROM CIM_Job
>> WHERE Priority > 1
>> '0'.get() | Format-Custom
>>

class ManagementObjectHost\cim2\Win32_PrintJob
{
    Document = Monod Manifesto - Public
    JobId = 6
    JobStatus =
    Owner = User
    Priority = 42
    Size = 1827888
    Name = Epson Stylus C960R 748 ESC/P 2, 6
}

PS C:\> $url = 'http://blogs.msdn.com/powershell/rss.aspx'
PS C:\> $obj = [xml](New-Object System.Net.WebClient).DownloadString($url)
PS C:\> $obj.rss.channel.item | select title -first 3

title
----
RSS: What's Coming In PowerShell V2
RSS Talk: System Center Foundation Technologies

PS C:\> $host.version.ToString().Insert(0, 'Windows PowerShell: ')
Windows PowerShell: 1.0.0.0
PS C:\>
```

Screenshot of a Windows PowerShell session

Paradigm	Imperative, pipeline, object-oriented, functional and reflective
Designed by	Jeffrey Snover, Bruce Payette, James Truher (et al.)
Developer	Microsoft
First appeared	November 14, 2006
Stable release	6.2.2 / July 16, 2019 ^[1]
Preview release	7.0.0-preview.3 / August 20, 2019 ^[2]
Typing discipline	Strong, safe, implicit and dynamic
Platform	.NET Framework, .NET Core
OS	Windows 7 and later Windows Server 2008 R2 and later macOS 10.12 and later

Design

Cmdlets
Pipeline
Scripting
Hosting

Desired State Configuration

Versions

PowerShell 1.0
PowerShell 2.0
PowerShell 3.0
PowerShell 4.0
PowerShell 5.0
PowerShell 5.1
PowerShell Core 6.0
PowerShell Core 6.1
PowerShell Core 6.2
PowerShell 7

Comparison of cmdlets with similar commands

File extensions

Application support

Snap-ins and hosts

Alternative implementation

See also

References

Further reading

External links

Ubuntu 14.04, 16.04, and 17.04

Debian 8.7+, and 9

CentOS 7

Red Hat Enterprise Linux 7

OpenSUSE 42.2

Fedora 25, 26

License

MIT License^[3] (but the Windows component remains proprietary)

Filename extensions

.ps1 (Script)
.ps1xml (XML Document)
.psc1 (Console File)
.psd1 (Data File)
.psm1 (Script Module)
.pssc (Session Configuration File)
.cdxml (Cmdlet Definition XML Document)

Website

microsoft.com/powershell (https://microsoft.com/powershell)

Influenced by

Python, Ksh, Perl, C#, CL, DCL, SQL, Tcl, Tk,^[4]
Chef, Puppet

Background

Every version of Microsoft Windows for personal computers has included a command line interpreter (CLI) for managing the operating system. Its predecessor, MS-DOS, relied exclusively on a CLI. These are COMMAND.COM in MS-DOS and Windows 9x, and cmd.exe in the Windows NT family of operating systems. Both support a few basic internal commands. For other purposes, a separate console application must be written. They also include a basic scripting language (batch files), which can be used to automate various tasks. However, they cannot be used to automate all facets of graphical user interface (GUI) functionality, in part because command-line equivalents of operations are limited, and the scripting language is elementary. In Windows Server 2003, the situation was improved, but scripting support was still unsatisfactory.^[10]

Microsoft attempted to address some of these shortcomings by introducing the Windows Script Host in 1998 with Windows 98, and its command-line based host: cscript.exe. It integrates with the Active Script engine and allows scripts to be written in compatible languages, such as JScript and VBScript, leveraging the APIs exposed by applications via COM. However, it has its own deficiencies: its documentation is not very accessible, and it quickly gained a reputation as a system vulnerability vector after several high-profile computer viruses exploited weaknesses in its security provisions. Different versions of Windows provided various special-purpose command line interpreters (such as netsh and WMIC) with their own command sets but they were not interoperable.

In an interview published 2017 September 13, Jeffrey Snover explained the motivation for the project:^[11]

|

I'd been driving a bunch of managing changes, and then I originally took the UNIX tools and made them available on Windows, and then it just didn't work. Right? Because there's a core architectural difference between Windows and Linux. On Linux, everything's an ASCII text file, so anything that can manipulate that is a managing tool. AWK, grep, sed? Happy days!

I brought those tools available on Windows, and then they didn't help manage Windows because in Windows, everything's an API that returns structured data. So, that didn't help. [...] I came up with this idea of PowerShell, and I said, "Hey, we can do this better."

By 2002 Microsoft had started to develop a new approach to command line management, including a CLI called Monad (also known as Microsoft Shell or MSH). The ideas behind it were published in August 2002 in a white paper titled Monad Manifesto.^[12] Monad was to be a new extensible CLI with a fresh design that would be capable of automating a full range of core administrative tasks. Microsoft first showed off Monad at the Professional Development Conference in Los Angeles in October 2003. A private beta program began a few months later which eventually led to a public beta program.

Microsoft published the first Monad public beta release on June 17, 2005, Beta 2 on September 11, 2005, and Beta 3 on January 10, 2006. Not much later, on April 25, 2006 Microsoft formally announced that Monad had been renamed *Windows PowerShell*, positioning it as a significant part of their management technology offerings.^[13] Release Candidate 1 of PowerShell was released at the same time. A significant aspect of both the name change and the RC was that this was now a component of Windows, and not an add-on product.

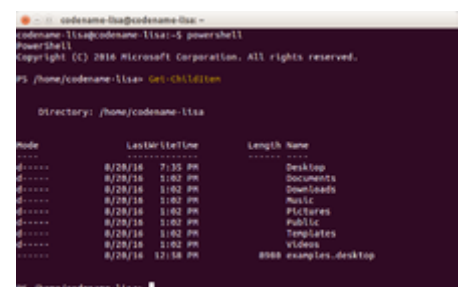
Release Candidate 2 of PowerShell version 1 was released on September 26, 2006 with final Release to the web (RTW) on November 14, 2006 and announced at TechEd Barcelona. PowerShell for earlier versions of Windows was released on January 30, 2007.^[14]

PowerShell v2.0 development began before PowerShell v1.0 shipped. During the development, Microsoft shipped three community technology previews (CTP). Microsoft made these releases available to the public. The last CTP release of Windows PowerShell v2.0 was made available in December 2008.

PowerShell v2.0 was completed and released to manufacturing in August 2009, as an integral part of Windows 7 and Windows Server 2008 R2. Versions of PowerShell for Windows XP, Windows Server 2003, Windows Vista and Windows Server 2008 were released in October 2009 and are available for download for both 32-bit and 64-bit platforms.^[15]

Windows 10 shipped a testing framework for PowerShell.^[16]

On 18 August 2016, Microsoft announced^[17] that they had made PowerShell open-source and cross-platform with support for Windows, macOS, CentOS and Ubuntu.^[5] The source code was published on GitHub.^[18] The move to open source created a second incarnation of PowerShell called "PowerShell Core", which runs on .NET Core. It is distinct from "Windows PowerShell", which runs on the full .NET Framework.^[19] Starting with version 5.1, PowerShell Core is bundled with Windows Server 2016 Nano Server.^{[20][21]}



PowerShell for Linux 6.0 Alpha 9 on Ubuntu 14.04 x64

Design

PowerShell's developers based the core grammar of the tool on that of the POSIX 1003.2 Korn shell.^[22]

Windows PowerShell can execute four kinds of named commands:^[23]

- *cmdlets* (.NET Framework programs designed to interact with PowerShell)
- PowerShell scripts (files suffixed by `.ps1`)
- PowerShell functions
- standalone executable programs

If a command is a standalone executable program, PowerShell launches it in a separate process; if it is a cmdlet, it executes in the PowerShell process. PowerShell provides an interactive command-line interface, wherein the commands can be entered and their output displayed. The user interface, based on the Win32 console, offers customizable tab completion. PowerShell enables the creation of aliases for cmdlets, which PowerShell textually translates into invocations of the original commands. PowerShell supports both named and positional parameters for commands. In executing a cmdlet, the job of binding the argument value to the parameter is done by PowerShell itself, but for external executables, arguments are parsed by the external executable independently of PowerShell interpretation.

The PowerShell *Extended Type System (ETS)* is based on the .NET type system, but with extended semantics (for example, propertySets and third-party extensibility). For example, it enables the creation of different views of objects by exposing only a subset of the data fields, properties, and methods, as well as specifying custom formatting and sorting behavior. These views are mapped to the original object using XML-based configuration files.^[24]

Cmdlets

Cmdlets are specialized commands in the PowerShell environment that implement specific functions. These are the native commands in the PowerShell stack. Cmdlets follow a *Verb-Noun* naming pattern, such as *Get-ChildItem*, helping to make them self-descriptive.^[25] Cmdlets output their results as objects and can also receive objects as input, making them suitable for use as recipients in a pipeline. If a cmdlet outputs multiple objects, each object in the collection is passed down through the entire pipeline before the next object is processed.^[25]

Cmdlets are specialized .NET classes, which the PowerShell runtime instantiates and invokes at run-time. Cmdlets derive either from `Cmdlet` or from `PSCmdlet`, the latter being used when the cmdlet needs to interact with the PowerShell runtime.^[25] These base classes specify certain methods – `BeginProcessing()`, `ProcessRecord()` and `EndProcessing()` – which the cmdlet's implementation overrides to provide the functionality. Whenever a cmdlet runs, PowerShell invokes these methods in sequence, with `ProcessRecord()` being called if it receives pipeline input.^[26] If a collection of objects is piped, the method is invoked for each object in the collection. The class implementing the `Cmdlet` must have one .NET attribute – `CmdletAttribute` – which specifies the verb and the noun that make up the name of the cmdlet. Common verbs are provided as an enum.^{[27][28]}

If a cmdlet receives either pipeline input or command-line parameter input, there must be a corresponding property in the class, with a mutator implementation. PowerShell invokes the mutator with the parameter value or pipeline input, which is saved by the mutator implementation in class variables. These values are then referred to by the methods which implement the functionality. Properties that map to command-line parameters are marked by `ParameterAttribute`^[29] and are set before the call to `BeginProcessing()`. Those which map to pipeline input are also flanked by `ParameterAttribute`, but with the `ValueFromPipeline` attribute parameter set.^[30]

The implementation of these cmdlet classes can refer to any .NET API and may be in any .NET language. In addition, PowerShell makes certain APIs available, such as `WriteObject()`, which is used to access PowerShell-specific functionality, such as writing resultant objects to the pipeline. Cmdlets can use .NET data access APIs directly or use the PowerShell infrastructure of PowerShell *Providers*, which make data stores addressable using unique paths. Data stores are exposed using drive letters, and hierarchies within them, addressed as directories. Windows PowerShell ships with providers for the file system, registry, the certificate store, as well as the namespaces for command aliases, variables, and functions.^[31] Windows PowerShell

also includes various cmdlets for managing various Windows systems, including the file system, or using Windows Management Instrumentation to control Windows components. Other applications can register cmdlets with PowerShell, thus allowing it to manage them, and, if they enclose any datastore (such as databases), they can add specific providers as well.

PowerShell V2 added a more portable version of Cmdlets called Modules. The PowerShell V2 release notes state:

Modules allow script developers and administrators to partition and organize their Windows PowerShell code in self-contained, reusable units. Code from a module executes in its own self-contained context and does not affect the state outside of the module. Modules also enable you to define a restricted runspace environment by using a script.^[32]

Pipeline

PowerShell implements the concept of a *pipeline*, which enables piping the output of one cmdlet to another cmdlet as input. For example, the output of the `Get-Process` cmdlet could be piped to the `Where-Object` to filter any process that has less than 1 MB of paged memory, and then to the `Sort-Object` cmdlet (e.g., to sort the objects by handle count), and then finally to the `Select-Object` cmdlet to select just the first 10 (i.e., the 10 processes based on handle count).

As with Unix pipelines, PowerShell pipelines can construct complex commands, using the `|` operator to connect stages. However, the PowerShell pipeline differs from Unix pipelines in that stages execute within the PowerShell runtime rather than as a set of processes coordinated by the operating system, and structured .NET objects, rather than byte streams, are passed from one stage to the next. Using objects and executing stages within the PowerShell runtime eliminates the need to serialize data structures, or to extract them by explicitly parsing text output.^[33] An object can also encapsulate certain functions that work on the contained data, which become available to the recipient command for use.^{[34][35]} For the last cmdlet in a pipeline, PowerShell automatically pipes its output object to the `Out-Default` cmdlet, which transforms the objects into a stream of format objects and then renders those to the screen.^{[36][37]}

Because all PowerShell objects are .NET objects, they share a `.ToString()` method, which retrieves the text representation of the data in an object. In addition, PowerShell allows formatting definitions to be specified, so the text representation of objects can be customized by choosing which data elements to display, and in what manner. However, in order to maintain backwards compatibility, if an external executable is used in a pipeline, it receives a text stream representing the object, instead of directly integrating with the PowerShell type system.^{[38][39][40]}

Scripting

Windows PowerShell includes a dynamically typed scripting language which can implement complex operations using cmdlets imperatively. The scripting language supports variables, functions, branching (if-then-else), loops (while, do, for, and foreach), structured error/exception handling and closures/lambda expressions,^[41] as well as integration with .NET. Variables in PowerShell scripts are prefixed with `$`. Variables can be assigned any value, including the output of cmdlets. Strings can be enclosed either in single quotes or in double quotes: when using double quotes, variables will be expanded even if they are inside the quotation marks. Enclosing the path to a file in braces preceded by a dollar sign (as in `${C:\foo.txt}`) creates a reference to the contents of the file. If it is used as an L-value, anything assigned to it will be written to the file. When used as an R-value, the contents of the file will be read. If an object is assigned, it is serialized before being stored.

Object members can be accessed using `.` notation, as in *C#* syntax. PowerShell provides special variables, such as `$args`, which is an array of all the command line arguments passed to a function from the command line, and `$_`, which refers to the current object in the pipeline.^[42] PowerShell also provides arrays and associative arrays. The PowerShell scripting language also

evaluates arithmetic expressions entered on the command line immediately, and it parses common abbreviations, such as GB, MB, and KB.^{[43][44]}

Using the `function` keyword, PowerShell provides for the creation of functions, the following general form:^[45]

```
function name ($Param1, $Param2)
{
    Instructions
}
```

The defined function is invoked in either of the following forms:^[45]

```
name value1 value2
name -Param1 value1 -Param2 value2
```

PowerShell supports named parameters, positional parameters, switch parameters and dynamic parameters.^[45]

PowerShell allows any .NET methods to be called by providing their namespaces enclosed in brackets ([]), and then using a pair of colons (::) to indicate the static method.^[46] For example, [System.Console]::WriteLine("PowerShell"). Objects are created using the `New-Object` cmdlet. Calling methods of .NET objects is accomplished by using the regular . notation.^[46]

PowerShell accepts strings, both raw and escaped. A string enclosed between single quotation marks is a raw string while a string enclosed between double quotation marks is an escaped string. PowerShell treats straight and curly quotes as equivalent.^[47]

The following list of special characters is supported by PowerShell.^[48]

PowerShell Special Characters

Character	Description
`0	<u>Null</u>
`a	<u>Alert</u>
`b	<u>Backspace</u>
`e	<u>Escape</u>
`f	<u>Form feed</u>
`n	<u>Newline</u>
`r	<u>Carriage return</u>
`t	<u>Horizontal tab</u>
`u{x}	<u>Unicode</u> escape sequence
`v	<u>Vertical tab</u>
--%	Stop parsing

For error handling, PowerShell provides a .NET-based exception-handling mechanism. In case of errors, objects containing information about the error (Exception object) are thrown, which are caught using the `try . . . catch` construct (although a `trap` construct is supported as well). PowerShell can be configured to silently resume execution, without actually throwing the exception; this can be done either on a single command, a single session or perpetually.^[49]

Scripts written using PowerShell can be made to persist across sessions in either a `.ps1` file or a `.psm1` file (the latter is used to implement a module). Later, either the entire script or individual functions in the script can be used. Scripts and functions operate analogously with cmdlets, in that they can be used as commands in pipelines, and parameters can be bound to them. Pipeline objects can be passed between functions, scripts, and cmdlets seamlessly. To prevent unintentional running of scripts, script execution is disabled by default and must be enabled explicitly.^[50] Enabling of scripts can be performed either at system, user or session level. PowerShell scripts can be signed to verify their integrity, and are subject to Code Access Security.^[51]

The PowerShell scripting language supports binary prefix notation similar to the scientific notation supported by many programming languages in the C-family.^[52]

Hosting

One can also use PowerShell embedded in a management application, which uses the PowerShell runtime to implement the management functionality. For this, PowerShell provides a managed hosting API. Via the APIs, the application can instantiate a *runspace* (one instantiation of the PowerShell runtime), which runs in the application's process and is exposed as a `Runspace` object.^[6] The state of the runspace is encased in a `SessionState` object. When the runspace is created, the Windows PowerShell runtime initializes the instantiation, including initializing the providers and enumerating the cmdlets, and updates the `SessionState` object accordingly. The `Runspace` then must be opened for either synchronous processing or asynchronous processing. After that it can be used to execute commands.

To execute a command, a pipeline (represented by a `Pipeline` object) must be created and associated with the runspace. The pipeline object is then populated with the cmdlets that make up the pipeline. For sequential operations (as in a PowerShell script), a `Pipeline` object is created for each statement and nested inside another `Pipeline` object.^[6] When a pipeline is created, Windows PowerShell invokes the pipeline processor, which resolves the cmdlets into their respective assemblies (the *command processor*) and adds a reference to them to the pipeline, and associates them with `InputPipe`, `OutputPipe` and `ErrorOutputPipe` objects, to represent the connection with the pipeline. The types are verified and parameters bound using reflection.^[6] Once the pipeline is set up, the host calls the `Invoke()` method to run the commands, or its asynchronous equivalent – `InvokeAsync()`. If the pipeline has the `Write-Host` cmdlet at the end of the pipeline, it writes the result onto the console screen. If not, the results are handed over to the host, which might either apply further processing or display the output itself.

Microsoft Exchange Server 2007 uses the hosting APIs to provide its management GUI. Each operation exposed in the GUI is mapped to a sequence of PowerShell commands (or pipelines). The host creates the pipeline and executes them. In fact, the interactive PowerShell console itself is a PowerShell host, which interprets the scripts entered at command line and creates the necessary `Pipeline` objects and invokes them.

Desired State Configuration

DSC allows for declaratively specifying how a software environment should be configured.^[53]

Upon running a *configuration*, DSC will ensure that the system gets the state described in the configuration. DSC configurations are idempotent. The *Local Configuration Manager* (LCM) periodically polls the system using the control flow described by *resources* (imperative pieces of DSC) to make sure that the state of a configuration is maintained.

Versions

Initially using the code name "Monad", PowerShell was first shown publicly at the Professional Developers Conference in September 2003. All major releases are still supported, and each major release has featured backwards compatibility with preceding versions.

PowerShell 1.0

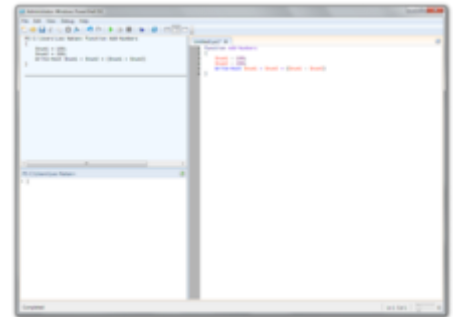
PowerShell 1.0 was released in November 2006 for [Windows XP SP2](#), [Windows Server 2003 SP1](#) and [Windows Vista](#).^[54] It is an optional component of [Windows Server 2008](#).

PowerShell 2.0

PowerShell 2.0 is integrated with [Windows 7](#) and [Windows Server 2008 R2](#)^[55] and is released for [Windows XP](#) with Service Pack 3, [Windows Server 2003](#) with Service Pack 2, and [Windows Vista](#) with Service Pack 1.^{[56][57]}

PowerShell v2 includes changes to the scripting language and hosting API, in addition to including more than 240 new cmdlets.^{[58][59]}

New features of PowerShell 2.0 include:^{[60][61][62]}



Windows PowerShell ISE v2.0 on Windows 7, an integrated development environment for PowerShell scripts.

- **PowerShell remoting:** Using [WS-Management](#), PowerShell 2.0 allows scripts and cmdlets to be invoked on a remote machine or a large set of remote machines.
- **Background jobs:** Also called a *PSJob*, it allows a command sequence (script) or pipeline to be invoked asynchronously. Jobs can be run on the local machine or on multiple remote machines. An interactive cmdlet in a PSJob blocks the execution of the job until user input is provided.
- **Transactions:** Enable cmdlet and developers can perform [transactional operations](#). PowerShell 2.0 includes transaction cmdlets for starting, committing, and rolling back a *PSTransaction* as well as features to manage and direct the transaction to the participating cmdlet and provider operations. The PowerShell Registry provider supports transactions.
- **Advanced functions:** These are cmdlets written using the PowerShell scripting language. Initially called "script cmdlets", this feature was later renamed "advanced functions".^[63]
- **Steppable Pipelines:** This allows the user to control when the `BeginProcessing()`, `ProcessRecord()` and `EndProcessing()` functions of a cmdlet are called.
- **Modules:** This allows script developers and administrators to organize and partition PowerShell scripts in self-contained, reusable units. Code from a [module](#) executes in its own self-contained context and does not affect the state outside the module. Modules can [define](#) a restricted runspace environment by using a script. They have a persistent state as well as public and private members.
- **Data language:** A domain-specific subset of the PowerShell scripting language that allows data definitions to be decoupled from the scripts and allows [localized](#) string resources to be imported into the script at runtime (*Script Internationalization*).
- **Script debugging:** It allows [breakpoints](#) to be set in a PowerShell script or function. Breakpoints can be set on lines, line & columns, commands and read or write access of variables. It includes a set of cmdlets to control the breakpoints via script.
- **Eventing:** This feature allows listening, forwarding, and acting on management and system events. Eventing allows PowerShell hosts to be notified about state changes to their managed entities. It also enables PowerShell scripts to subscribe to *ObjectEvents*, *PSEvents*, and *WmiEvents* and process them synchronously and asynchronously.
- **Windows PowerShell Integrated Scripting Environment (ISE):** PowerShell 2.0 includes a GUI-based PowerShell host that provides integrated debugger, [syntax highlighting](#), tab completion and up to 8 PowerShell Unicode-enabled consoles (Runspaces) in a tabbed UI, as well as the ability to run only the selected parts in a script.
- **Network file transfer:** Native support for prioritized, throttled, and asynchronous transfer of files between machines using the [Background Intelligent Transfer Service \(BITS\)](#).^[64]
- **New cmdlets:** Including `Out-GridView`, which displays tabular data in the [WPF GridView](#) object, on systems that allow it, and if ISE is installed and enabled.
- **New operators:** `-Split`, `-Join`, and `Splatting (@)` operators.
- **Exception handling with Try-Catch-Finally:** Unlike other .NET languages, this allows multiple exception types for a single catch block.
- **Nestable Here-Strings:** PowerShell [Here-Strings](#) have been improved and can now nest.^[65]

- **Block comments:** PowerShell 2.0 supports block comments using `<#` and `#>` as delimiters.^[66]
- **New APIs:** The new APIs range from handing more control over the PowerShell parser and runtime to the host, to creating and managing collection of Runspaces (*RunspacePools*) as well as the ability to create *Restricted Runspaces* which only allow a configured subset of PowerShell to be invoked. The new APIs also support participation in a Windows PowerShell managed transaction.

PowerShell 3.0

PowerShell 3.0 is integrated with Windows 8 and with Windows Server 2012. Microsoft has also made PowerShell 3.0 available for Windows 7 with Service Pack 1, for Windows Server 2008 with Service Pack 1, and for Windows Server 2008 R2 with Service Pack 1.^{[67][68]}

PowerShell 3.0 is part of a larger package, Windows Management Framework 3.0 (WMF3), which also contains the WinRM service to support remoting.^[68] Microsoft made several Community Technology Preview releases of WMF3. An early community technology preview 2 (CTP 2) version of Windows Management Framework 3.0 was released on 2 December 2011.^[69] Windows Management Framework 3.0 was released for general availability in December 2012^[70] and is included with Windows 8 and Windows Server 2012 by default.^[71]

New features in PowerShell 3.0 include:^{[68][72]:33–34}

- **Scheduled jobs:** Jobs can be scheduled to run on a preset time and date.
- **Session connectivity:** Sessions can be disconnected and reconnected. Remote sessions have become more tolerant of temporary network failures.
- **Improved code writing:** Code completion (IntelliSense) and snippets are added. PowerShell ISE allows users to use dialog boxes to fill in parameters for PowerShell cmdlets.
- **Delegation support:** Administrative tasks can be delegated to users who do not have permissions for that type of task, without granting them perpetual additional permissions.
- **Help update:** Help documentations can be updated via Update-Help command.
- **Automatic module detection:** Modules are loaded implicitly whenever a command from that module is invoked. Code completion works for unloaded modules as well.
- **New commands:** Dozens of new modules were added, including functionality to manage disks `get -WmiObject win32_logicaldisk`, volumes, firewalls, network connections and printer management, previously performed via WMI.

PowerShell 4.0

PowerShell 4.0 is integrated with Windows 8.1 and with Windows Server 2012 R2. Microsoft has also made PowerShell 4.0 available for Windows 7 SP1, Windows Server 2008 R2 SP1 and Windows Server 2012.^[73]

New features in PowerShell 4.0 include:

- **Desired State Configuration:**^{[74][75][76]} Declarative language extensions and tools that enable the deployment and management of configuration data for systems using the DMTF management standards and WS-Management Protocol
- **New default execution policy:** On Windows Servers, the default execution Policy is now RemoteSigned.
- **Save-Help:** Help can now be saved for modules that are installed on remote computers.
- **Enhanced debugging:** The debugger now supports debugging workflows, remote script execution and preserving debugging sessions across PowerShell session reconnections.
- **-PipelineVariable switch:** A new ubiquitous parameter to expose the current pipeline object as a variable for programming purposes
- **Network diagnostics** to manage physical and Hyper-V's virtualized network switches
- **Where and ForEach** method syntax provides an alternate method of filtering and iterating over objects.

PowerShell 5.0

Windows Management Framework (WMF) 5.0 RTM which includes PowerShell 5.0 was re-released to web on February 24, 2016, following an initial release with a severe bug.^[77] Key features include OneGet PowerShell cmdlets to support Chocolatey's repository-based package management^[78] and extending support for switch management to layer 2 network switches.^[79]



PowerShell 5.0 icon

New features in PowerShell 5.0 include:

- PowerShell class definitions (properties, methods)
- PowerShell .NET Enumerations
- Debugging for PowerShell Runspaces in remote processes
- Debugging for PowerShell Background Jobs
- Desired State Configuration (DSC) Local Configuration Manager (LCM) version 2.0
- DSC partial configurations
- DSC Local Configuration Manager meta-configurations
- Authoring of DSC resources using PowerShell classes

PowerShell 5.1

It was released along with the Windows 10 Anniversary Update^[80] on August 2, 2016, and in Windows Server 2016.^[81] PackageManagement now supports proxies, PSReadLine now has ViMode support, and two new cmdlets were added: Get-TimeZone and Set-TimeZone. The LocalAccounts module allows for adding/removing local user accounts.^[82] A preview for PowerShell 5.1 was released for Windows 7, Windows Server 2008, Windows Server 2008 R2, Windows Server 2012, and Windows Server 2012 R2 on July 16, 2016,^[83] and was released on January 19, 2017.^[84]

PowerShell 5.1 is the first version to come in two editions of "Desktop" and "Core". The "Desktop" edition is the continuation of the traditional Windows PowerShell that runs on full .NET Framework stack. The "Core" edition runs on .NET Core and is bundled with Windows Server 2016 Nano Server. In exchange for smaller footprint, the latter lacks some features such as the cmdlets to manage clipboard or join a computer to a domain, WMI version 1 cmdlets, Event Log cmdlets and profiles.^[21] This was the final version of PowerShell made exclusively for Windows.

PowerShell Core 6.0

PowerShell Core 6.0 was first announced on 18 August 2016, when Microsoft unveiled PowerShell Core and its decision to make the product cross-platform, independent of Windows, free and open source.^[5] It achieved general availability on 10 January 2018 for Windows, macOS and Linux.^[85] It has its own support lifecycle and adheres to the Microsoft lifecycle policy that is introduced with Windows 10: Only the latest version of PowerShell Core is supported. Microsoft expects to release one minor version for PowerShell Core 6.0 every six months.^[86]

The most significant change in this version of PowerShell is the expansion to the other platforms. For Windows administrators, this version of PowerShell is devoid of any major new features. In an interview with the community on 11 January 2018, the PowerShell team was asked to list the top 10 most exciting things that would happen for a Windows IT professional who would migrate from Windows PowerShell 5.1 to PowerShell Core 6.0; in response, Angel Calvo of Microsoft could only name two: cross-platform and open-source.^[87]

PowerShell Core 6.1

- Compatibility with 1900+ existing cmdlets in Windows 10 and Windows Server 2019^[88]
- Built on top of .NET Core 2.1

- Support for the latest versions of Windows, macOS, and Linux
- Significant performance improvements
- Markdown cmdlets
- Experimental feature flags

PowerShell Core 6.2

The PowerShell Core 6.2 release is focused primarily on performance improvements, bug fixes, and smaller cmdlet/language enhancements that improve the quality of life for users.^[89]

PowerShell 7

PowerShell 7 is intended to become the replacement product for PowerShell Core 6.x products as well as Windows PowerShell 5.1, which is the last supported Windows PowerShell version.^{[90][91]} In order for PowerShell 7 to be a viable replacement for Windows PowerShell 5.1 it must have near parity with Windows PowerShell in terms of compatibility with modules that ship with Windows.^[92]

Comparison of cmdlets with similar commands

The following table contains a selection of the cmdlets that ship with PowerShell, noting similar commands in other well-known command-line interpreters. Many of these similar commands come out-of-the-box defined as aliases within PowerShell, making it easy for people familiar with other common shells to start working.

Comparison of PowerShell cmdlets with internal and external commands of other command-line interpreters

PowerShell (Cmdlet)	PowerShell (Alias)	Windows Command Prompt	Unix shell	Description
Get-ChildItem	gci, dir, ls	<u>dir</u>	<u>ls</u>	Lists all files and folders in the current or given folder
Test-Connection ^[a]	<u>ping</u>	<u>ping</u>	<u>ping</u>	Sends ICMP echo requests to the specified machine from the current machine, or instructs another machine to do so
Get-Content	gc, type, cat	<u>type</u>	<u>cat</u>	Gets the content of a file
Get-Command	gcm	<u>help</u>	<u>type</u> , <u>which</u> , <u>compgen</u>	Lists available commands
Get-Help	help, man	<u>help</u>	<u>apropos</u> , <u>man</u>	Prints a command's documentation on the console
Clear-Host	cls, clear	<u>cls</u>	<u>clear</u>	Clears the screen ^[b]
Copy-Item	cpi, copy, cp	<u>copy</u> , <u>xcopy</u> , <u>robocopy</u>	<u>cp</u>	Copies files and folders to another location
Move-Item	mi, move, mv	<u>move</u>	<u>mv</u>	Moves files and folders to a new location
Remove-Item	ri, del, erase, rmdir, rd, rm	<u>del</u> , <u>erase</u> , <u>rmdir</u> , <u>rd</u>	<u>rm</u> , rmdir	Deletes files or folders
Rename-Item	rni, ren, mv	<u>ren</u> , <u>rename</u>	<u>mv</u>	Renames a single file, folder, hard link or symbolic link
Get-Location	gl, cd, pwd	<u>cd</u>	<u>pwd</u>	Displays the working path (current folder)
Pop-Location	popd	<u>popd</u>	popd	Changes the working path to the location most recently pushed onto the stack
Push-Location	pushd	<u>pushd</u>	pushd	Stores the working path onto the stack
Set-Location	sl, cd, chdir	<u>cd</u> , <u>chdir</u>	cd	Changes the working path
Tee-Object	tee	N/A	<u>tee</u>	Pipes input to a file or variable, passing the input along the pipeline
Write-Output	echo, write	<u>echo</u>	echo	Prints strings or other objects to the <u>standard output</u>
Get-Process	gps, ps	<u>tlist</u> , ^[c] <u>tasklist</u> ^[d]	<u>ps</u>	Lists all running processes
Stop-Process	spps, kill	<u>kill</u> , ^[c] <u>taskkill</u> ^[d]	kill ^[e]	Stops a running process
Select-String	sls	<u>findstr</u>	<u>find</u> , <u>grep</u>	Prints lines matching a pattern
Set-Variable	sv, set	<u>set</u>	env, export, set, setenv	Creates or alters the contents of an <u>environment variable</u>
Invoke-WebRequest	iwr, curl , wget ^[f]	<u>curl</u>	<u>wget</u> , curl	Gets contents from a web page on the Internet

Notes

- a. While the external ping command remains available to PowerShell, Test-Connection's output is a structured object that can be programmatically inspected.^[93]
- b. Clear-Host is implemented as a predefined PowerShell function.
- c. Available in Windows NT4, Windows 98 Resource Kit, Windows 2000 Support Tools
- d. Available in Windows XP Professional Edition and later
- e. Also used in UNIX to send a process any signal, the "Terminate" signal is merely the default
- f. curl and wget aliases are absent from PowerShell Core, so as to not interfere with invoking similarly named native commands.

File extensions

- PS1 – Windows PowerShell shell script^[94]
- PSD1 – Windows PowerShell data file (for Version 2)^[95]
- PSM1 – Windows PowerShell module file (for Version 2)^[96]
- PS1XML – Windows PowerShell format and type definitions^{[40][97]}
- CLIXML – Windows PowerShell serialized data^[98]
- PSC1 – Windows PowerShell console file^[99]
- PSSC – Windows PowerShell Session Configuration file^[100]

Application support

Snap-ins and hosts

Application	Version	Cmdlets	Provider	Management GUI
Exchange Server	2007	402	Yes	Yes
Windows Server	2008	Yes	Yes	No
Microsoft SQL Server	2008	Yes	Yes	No
Microsoft SharePoint	2010	Yes	Yes	No
System Center Configuration Manager	2012 R2	400+	Yes	No
System Center Operations Manager	2007	74	Yes	No
System Center Virtual Machine Manager	2007	Yes	Yes	Yes
System Center Data Protection Manager	2007	Yes	No	No
Windows Compute Cluster Server	2007	Yes	Yes	No
Microsoft Transporter Suite for Lotus Domino ^[101]	08.02.0012	47	No	No
Microsoft PowerTools for Open XML ^[102]	1.0	33	No	No
IBM WebSphere MQ ^[103]	6.0.2.2	44	No	No
Quest Management Shell for Active Directory ^[104]	1.7	95	No	No
Special Operations Software Specops Command ^[105]	1.0	Yes	No	Yes
VMware vSphere PowerCLI ^[106]	6.5 R1	500+	Yes	Yes
Internet Information Services ^[107]	7.0	54	Yes	No
Windows 7 Troubleshooting Center ^[108]	6.1	Yes	No	Yes
Microsoft Deployment Toolkit ^[109]	2010	Yes	Yes	Yes
NetApp PowerShell Toolkit ^{[110][111]}	4.2	2000+	Yes	Yes
JAMS Scheduler – Job Access & Management System ^[112]	5.0	52	Yes	Yes
UIAutomation ^[113]	0.8	432	No	No
Dell Equallogic ^[114]	3.5	55	No	No
LOGINventory ^[115]	5.8	Yes	Yes	Yes
SePSX ^[116]	0.4.1	39	No	No

Alternative implementation

A project named *Pash* (the name is a pun on the well-known "bash" Unix shell^[117]) has been an [open source](#) and [cross-platform](#) re-implementation of PowerShell via the [Mono framework](#). Pash was created by Igor Moochnick, written in [C#](#) and was released under the [GNU General Public License](#). Pash development stalled in 2008,^[117] was restarted on GitHub in 2012.^[118]

See also

- [Common Information Model](#)
- [Comparison of command shells](#)
- [Comparison of programming languages](#)
- [Web-Based Enterprise Management](#)
- [Windows Script Host](#)

- Windows Terminal

References

1. "Release v6.2.2 Release of PowerShell Core" (<https://github.com/PowerShell/PowerShell/releases/tag/v6.2.2>). *GitHub PowerShell Core repository*. Retrieved 2019-07-18.
2. "Release v7.0.0-preview.3 Release of PowerShell Core" (<https://github.com/PowerShell/PowerShell/releases/tag/v7.0.0-preview.3>). *GitHub PowerShell Core repository*. Retrieved 2019-08-21.
3. "PowerShell for every system!" (<https://github.com/PowerShell/PowerShell>). 12 June 2017 – via GitHub.
4. Snover, Jeffrey (May 25, 2008). "PowerShell and WPF: WTF" (<https://blogs.msdn.microsoft.com/powershell/2008/05/25/powershell-and-wpf-wtf/>). *Windows PowerShell Blog*. Microsoft.
5. Bright, Peter (18 August 2016). "PowerShell is Microsoft's latest open source release, coming to Linux, OS X" (<https://arstechnica.com/information-technology/2016/08/powershell-is-microsofts-latest-open-source-release-coming-to-linux-os-x/>). *Ars Technica*. Condé Nast.
6. "How Windows PowerShell works" (<http://msdn2.microsoft.com/en-us/library/ms714658.aspx>). *Microsoft Developer Network*. Microsoft. Retrieved 2007-11-27.
7. Truher, Jim (December 2007). "Extend Windows PowerShell With Custom Commands" (<https://web.archive.org/web/20081006195551/http://msdn.microsoft.com/en-us/magazine/cc163293.aspx>). *MSDN Magazine*. Microsoft. Archived from the original (<https://msdn.microsoft.com/en-us/magazine/cc163293.aspx>) on 6 October 2008.
8. Lowe, Scott (January 4, 2007). "Exchange 2007: Get used to the command line" (<https://www.techrepublic.com/article/exchange-2007-get-used-to-the-command-line/>). *TechRepublic*. CBS Interactive.
9. Snover, Jeffrey (November 13, 2007). "SQL Server Support for PowerShell!" (<http://blogs.msdn.com/powershell/archive/2007/11/13/sql-server-support-for-powershell.aspx>). *Windows PowerShell Blog* (blog posting). Microsoft.
10. Dragan, Richard V. (April 23, 2003). "Windows Server 2003 Delivers Improvements All Around" (<https://www.pcmag.com/article2/0,2704,1040410,00.asp>). Reviews. *PC Magazine*. Ziff Davis. "A standout feature here is that virtually all admin utilities now work from the command line (and most are available through telnet)."
11. Biggar and Harbaugh (2017-09-14). "The Man Behind Windows PowerShell" (<https://www.heavybit.com/library/podcasts/to-be-continuous/ep-37-the-man-behind-windows-powershell/>). *To Be Continuous* (Podcast). Heavybit. Retrieved 2017-09-14.
12. Snover, Jeffrey (August 2, 2002). "Monad Manifesto – the Origin of Windows PowerShell" (<http://blogs.msdn.com/b/powershell/archive/2007/03/19/monad-manifesto-the-origin-of-windows-powershell.aspx>). *Windows PowerShell Blog* (blog posting). Microsoft.
13. "Windows PowerShell (Monad) Has Arrived" (<https://blogs.msdn.microsoft.com/powershell/2006/04/25/windows-powershell-m Monad-has-arrived/>). *Windows PowerShell Blog*. Microsoft. April 25, 2006.
14. Snover, Jeffrey (November 15, 2006). "Windows PowerShell & Windows Vista" (<http://blogs.msdn.com/powershell/archive/2006/11/15/windows-powershell-windows-vista.aspx>). *Windows PowerShell Blog* (blog posting). Microsoft.
15. "Windows Management Framework (Windows PowerShell 2.0, WinRM 2.0, and BITS 4.0)" (<https://web.archive.org/web/20131013100052/http://support.microsoft.com/kb/968929>). *Support*. Microsoft. September 30, 2013. Archived from the original (<http://support.microsoft.com/kb/968929>) on October 13, 2013.
16. "What is Pester and Why Should I Care?" (<https://blogs.technet.microsoft.com/heyscriptingguy/2015/12/14/what-is-pester-and-why-should-i-care/>).
17. Snover, Jeffrey (18 August 2016). "PowerShell is open sourced and is available on Linux" (<https://azure.microsoft.com/en-us/blog/powershell-is-open-sourced-and-is-available-on-linux/>). *Microsoft Azure Blog*. Microsoft.
18. "PowerShell/PowerShell" (<https://github.com/PowerShell/PowerShell>). *GitHub*. Retrieved 2016-08-18.
19. Hansen, Kenneth; Calvo, Angel (August 18, 2016). "PowerShell on Linux and Open Source!" (<https://blogs.msdn.microsoft.com/powershell/2016/08/18/powershell-on-linux-and-open-source-2/>). *Windows PowerShell Blog*. Microsoft.

20. Foley, Mary Jo (August 18, 2016). "Microsoft open sources PowerShell; brings it to Linux and Mac OS X" (<https://www.zdnet.com/article/microsoft-open-sources-powershell-brings-it-to-linux-and-mac-os-x/>). *ZDNet*. CBS Interactive.
21. "PowerShell on Nano Server" (<https://technet.microsoft.com/en-us/windows-server-docs/get-started/powershell-on-nano-server>). *TechNet*. Microsoft. 20 October 2016.
22. Payette, Bruce (2007). *Windows PowerShell in Action* (<https://books.google.com/books?id=MYZQAAAAMAAJ>). Manning Pubs Co Series. Manning. p. 27. ISBN 9781932394900. Retrieved 2016-07-22. "The core PowerShell language is based on the POSIX 1003.2 grammar for the Korn shell."
23. "about_Command_Precedence" (<https://technet.microsoft.com/en-us/library/hh848304.aspx>). *TechNet*. Microsoft. May 8, 2014.
24. "Windows PowerShell Extended Type System" (<http://msdn2.microsoft.com/en-us/library/ms714419.aspx>). Retrieved 2007-11-28.
25. "Windows PowerShell Cmdlets" (<http://msdn2.microsoft.com/en-us/library/ms714395.aspx>). Retrieved 2007-11-28.
26. "Creating Your First Cmdlet" (<http://msdn2.microsoft.com/en-us/library/ms714622.aspx>). Retrieved 2007-11-28.
27. "Get-Verb" (<https://technet.microsoft.com/en-us/library/hh852690.aspx>). *TechNet*. Microsoft. May 8, 2014.
28. "Cmdlet Overview" (<https://msdn.microsoft.com/en-us/library/ms714395%28v=vs.85%29.aspx>). *MSDN*. Microsoft. May 8, 2014.
29. "Adding parameters That Process Command Line Input" (<http://msdn2.microsoft.com/en-us/library/ms714663.aspx>). Retrieved 2007-11-28.
30. "Adding parameters That Process Pipeline Input" (<http://msdn2.microsoft.com/en-us/library/ms714597.aspx>). Retrieved 2007-11-28.
31. "Windows PowerShell Providers" (<https://technet.microsoft.com/en-us/library/dd347723.aspx>). Retrieved 2010-10-14.
32. PowerShell V2 release notes
33. "Windows PowerShell Owner's Manual: Piping and the Pipeline in Windows PowerShell" (<https://technet.microsoft.com/en-us/library/ee176927.aspx>). *TechNet*. Microsoft. Retrieved 2011-09-27.
34. Jones, Don (2008). "Windows PowerShell – Rethinking the Pipeline" (<http://www.microsoft.com/technet/technetmag/issues/2007/07/PowerShell/default.aspx>). *Microsoft TechNet*. Microsoft. Retrieved 2007-11-28.
35. "Windows PowerShell Object Concepts" (<https://web.archive.org/web/20070819233213/http://msdn2.microsoft.com/en-us/library/aa347685.aspx>). Archived from the original (<http://msdn2.microsoft.com/en-us/library/aa347685.aspx>) on August 19, 2007. Retrieved 2007-11-28.
36. "How PowerShell Formatting and Outputting REALLY works" (<http://blogs.msdn.com/powershell/archive/2006/04/30/586973.aspx>). Retrieved 2007-11-28.
37. "More – How does PowerShell formatting really work?" (<http://blogs.msdn.com/powershell/archive/2006/06/21/641738.aspx>). Retrieved 2007-11-28.
38. "about_Pipelines" (<https://technet.microsoft.com/en-us/library/hh847902.aspx>). *TechNet*. Microsoft. May 8, 2014.
39. "about_Objects" (<https://technet.microsoft.com/en-us/library/hh847810.aspx>). *TechNet*. Microsoft. May 8, 2014.
40. "about_Format.ps1xml" (<https://technet.microsoft.com/en-us/library/hh847831.aspx>). *TechNet*. Microsoft. May 8, 2014.
41. "Anonymous Functions and Code Blocks in PowerShell" (<http://defndo.com/powershell-code-blocks-and-anonymous-functions/>). Retrieved 2012-01-21.
42. "Introduction to Windows PowerShell's Variables" (http://www.computerperformance.co.uk/powershell/powershell_variables.htm). Retrieved 2007-11-28.
43. "Byte Conversion" (<https://technet.microsoft.com/en-us/library/ee692684.aspx>). *Windows PowerShell Tip of the Week*. Retrieved 15 November 2013.
44. Ravikanth (20 May 2013). "Converting to size units (KB, MB,GB,TB, and PB) without using PowerShell multipliers" (<http://www.powershellmagazine.com/2013/05/20/converting-to-size-units-kb-mbgbtb-and-pb-without-using-powershell-multipliers/>). *PowerShell Magazine*.

45. "about_Functions" (<https://technet.microsoft.com/en-us/library/hh847829.aspx>). *Microsoft TechNet*. Microsoft. 17 October 2013. Retrieved 15 November 2013.
46. "Lightweight Testing with Windows PowerShell" (<http://msdn.microsoft.com/msdnmag/issues/07/05/TestRun/default.aspx>). Retrieved 2007-11-28.
47. Angelopoulos, Alex; Karen, Bemowski (4 December 2007). "PowerShell Got Smart About Smart Quotes" (<http://windowsitpro.com/powershell/powershell-got-smart-about-smart-quotes>). *Windows IT Pro*. Penton Media. Retrieved 15 November 2013.
48. "About Special Characters" (https://docs.microsoft.com/en-us/powershell/module/microsoft.powershell.core/about/about_special_characters?view=powershell-6). PowerShell / Scripting. Microsoft. June 8, 2017. Retrieved June 20, 2019.
49. "Trap [Exception] { "In PowerShell" }" (<http://huddledmasses.org/trap-exception-in-powershell>). Retrieved 2007-11-28.
50. "Running Windows PowerShell Scripts" (<http://www.microsoft.com/technet/scriptcenter/topics/winpsch/manual/run.msp>). Retrieved 2007-11-28.
51. "about_Signing" (<https://technet.microsoft.com/en-us/library/hh847874.aspx>). *Microsoft TechNet*. Microsoft. 17 October 2013. Retrieved 15 November 2013.
52. Lee Holmes (September 2006). *Windows PowerShell Quick Reference*. O'Reilly Media.
53. eslesar. "Windows PowerShell Desired State Configuration Overview" (<https://msdn.microsoft.com/en-us/powershell/dsc/overview>). *msdn.microsoft.com*.
54. Chung, Leonard; Snover, Jeffrey; Kumaravel, Arul (14 November 2006). "It's a Wrap! Windows PowerShell 1.0 Released!" (<http://blogs.msdn.com/b/powershell/archive/2006/11/14/windows-powershell-1-0-released.aspx>). *Windows PowerShell Blog*. Microsoft.
55. "PowerShell will be installed by default on Windows Server 08 R2 (WS08R2) and Windows 7 (W7)!" (<http://blogs.msdn.com/powershell/archive/2008/10/28/powershell-will-be-installed-by-default-on-windows-server-08-r2-ws08r2-and-windows-7-w7.aspx>). *Windows PowerShell Blog*. Microsoft. 2008-10-28. Retrieved 2011-09-27.
56. "Windows Management Framework is here!" (<http://blogs.msdn.com/powershell/archive/2009/10/27/windows-management-framework-is-here.aspx>). 2009-10-27. Retrieved 2009-10-30.
57. "Microsoft Support Knowledge Base: Windows Management Framework (Windows PowerShell 2.0, WinRM 2.0, and BITS 4.0)" (<http://support.microsoft.com/kb/968929>). Support.microsoft.com. 2011-09-23. Retrieved 2011-09-27.
58. "574 Reasons Why We Are So Proud and Optimistic About W7 and WS08R2" (<http://blogs.msdn.com/powershell/archive/2008/10/29/574-reasons-why-we-are-so-proud-and-optimistic-about-w7-and-ws08r2.aspx>). *Windows PowerShell Blog*. Microsoft. 2008-10-29. Retrieved 2011-09-27.
59. Snover, Jeffrey (2008). "PowerShell: Creating Manageable Web Services" (<https://web.archive.org/web/20081013065033/http://channel9.msdn.com/pdc2008/ES24/>). Archived from the original (<http://channel9.msdn.com/pdc2008/ES24/>) on October 13, 2008. Retrieved July 19, 2015.
60. "What's New in CTP of PowerShell 2.0" (<http://blogs.msdn.com/powershell/archive/2007/11/06/what-s-new-in-ctp-of-powershell-2-0.aspx>). Retrieved 2007-11-28.
61. "Windows PowerShell V2 Community Technology Preview 2 (CTP2) – releaseNotes" (<https://web.archive.org/web/20080506150324/http://www.microsoft.com/downloads/details.aspx?FamilyID=7c8051c2-9bfc-4c81-859d-0864979fa403&DisplayLang=en>). Archived from the original (<http://www.microsoft.com/downloads/details.aspx?FamilyID=7C8051C2-9BFC-4C81-859D-0864979FA403&displaylang=en>) on May 6, 2008. Retrieved 2008-05-05.
62. "Differences between PowerShell 1.0 and PowerShell 2.0" (<http://activexperts.com/admin/powershell/ps1vs2/>). Retrieved 2010-06-26.
63. Jones, Don (May 2010). "Windows PowerShell: Writing Cmdlets in Script" (<https://technet.microsoft.com/en-us/library/ff677563.aspx>). *TechNet Magazine*. Microsoft.
64. "GoGrid Snap-in – Managing Cloud Services with PowerShell" (<http://blogs.msdn.com/powershell/archive/2008/10/14/gogrid-snap-in-managing-cloud-services-with-powershell.aspx>). *Windows PowerShell Blog*. Microsoft. 2008-10-14. Retrieved 2011-09-27.
65. "Emit-XML" (<http://blogs.msdn.com/powershell/archive/2008/10/18/emit-xml.aspx>). *Windows PowerShell Blog*. Microsoft. 2008-10-17. Retrieved 2011-09-27.

66. "Block Comments in V2" (<http://blogs.msdn.com/powershell/archive/2008/06/14/block-comments-in-v2.aspx>). *Windows PowerShell Blog*. Microsoft. 2008-06-14. Retrieved 2011-09-27.
67. Lee, Thomas (13 August 2012). "PowerShell Version 3 is RTM!" (<http://tfl09.blogspot.co.uk/2012/08/powershell-v3-is-rtm.html>). *Under The Stairs*. Retrieved 2012-08-13.
68. "Windows Management Framework 3.0" (<http://www.microsoft.com/en-us/download/details.aspx?id=34595>). *Download Center*. Microsoft. 4 September 2012. Retrieved 2012-11-08.
69. "Windows Management Framework 3.0 Community Technology Preview (CTP) #2 Available for Download" (<http://blogs.msdn.com/b/powershell/archive/2011/12/02/windows-management-framework-3-0-community-technology-preview-ctp-2-available-for-download.aspx>). *Windows PowerShell Blog*. Microsoft. 2 December 2011.
70. "Windows Management Framework 3.0" (<https://www.microsoft.com/en-us/download/details.aspx?id=34595>). *Download Center*. Microsoft. 3 December 2012.
71. Jofre, JuanPablo (December 14, 2016). "Windows PowerShell System Requirements" (<https://msdn.microsoft.com/en-us/powershell/scripting/setup/windows-powershell-system-requirements>). *Microsoft Developer Network*. Microsoft. Retrieved April 20, 2017.
72. Honeycutt, Jerry (2012). Woolley, Valerie (ed.). *Introducing Windows 8: An Overview for IT Professionals* (http://blogs.msdn.com/b/microsoft_press/archive/2012/11/13/free-ebook-introducing-windows-8-an-overview-for-it-professionals-final-edition.aspx). Redmond, WA: Microsoft Press. ISBN 978-0-7356-7050-1.
73. "Windows Management Framework 4.0 is now available" (<http://blogs.msdn.com/b/powershell/archive/2013/10/25/windows-management-framework-4-0-is-now-available.aspx>). Microsoft. 24 October 2013. Retrieved 4 November 2013.
74. Levy, Shay (25 June 2013). "New Features in Windows PowerShell 4.0" (<http://www.powershellmagazine.com/2013/06/25/new-features-in-windows-powershell-4-0/>). *PowerShell Magazine*. Retrieved 26 June 2013.
75. "Desired State Configuration in Windows Server 2012 R2 PowerShell" (<http://channel9.msdn.com/Events/TechEd/NorthAmerica/2013/MDC-B302#fbid=sBK5uHH2bcL>). *Channel 9*. Microsoft. 3 June 2013. Retrieved 26 June 2013.
76. Hall, Adrian (7 June 2013). "Thoughts from Microsoft TechEd North America" (<http://blogs.splunk.com/2013/06/07/thoughts-from-microsoft-teched-north-america/>). *Blogs: Tips & Tricks*. Splunk. Retrieved 26 June 2013.
77. "Windows Management Framework (WMF) 5.0 RTM packages has been republished" (<https://blogs.msdn.microsoft.com/powershell/2016/02/24/windows-management-framework-wmf-5-0-rtm-packages-has-been-republished/>). *Windows PowerShell Blog*. Microsoft. February 24, 2016.
78. "Q and A" (<https://github.com/OneGet/oneget/wiki/Q-and-A>). *GitHub*. Retrieved 21 April 2015.
79. Snover, Jeffrey (2014-04-03). "Windows Management Framework V5 Preview" (<http://blogs.technet.com/b/windowsserver/archive/2014/04/03/windows-management-framework-v5-preview.aspx>). *blogs.technet.com*. Microsoft. Retrieved 2015-04-21.
80. says, Jaap Brasser (2 August 2016). "#PSTip New PowerShell Commands in Windows 10 Anniversary Update" (<http://www.powershellmagazine.com/2016/08/02/pstip-new-powershell-commands-in-windows-10-anniversary-update/>).
81. "What's New In Windows Server 2016 Standard Edition Part 9 – Management And Automation" (<https://blogs.technet.microsoft.com/auoemteam/2016/09/04/whats-new-in-windows-server-2016-standard-edition-part-9-management-and-automation/>).
82. "Microsoft.PowerShell.LocalAccounts Module" (<https://technet.microsoft.com/en-us/library/mt651681.aspx>). *technet.microsoft.com*.
83. "Announcing Windows Management Framework (WMF) 5.1 Preview" (<https://blogs.msdn.microsoft.com/powershell/2016/07/16/announcing-windows-management-framework-wmf-5-1-preview/>).
84. "WMF 5.1" (<https://www.microsoft.com/en-us/download/details.aspx?id=54616>). *Microsoft Download Center*.
85. Aiello, Joey (11 January 2018). "PowerShell Core 6.0: Generally Available (GA) and Supported!" (<https://archive.is/nBkqv>). *PowerShell Team Blog*. Microsoft. Archived from the original (<https://blogs.msdn.microsoft.com/powershell/2018/01/10/powershell-core-6-0-generally-available-ga-and-supported/>) on 11 June 2018. Retrieved 11 June 2018.
86. Aiello, Joey; Wheeler, Sean (10 January 2018). "PowerShell Core Support Lifecycle" (<https://docs.microsoft.com/en-us/powershell/scripting/PowerShell-Core-Support?view=powershell-6>). *Microsoft Docs*. Microsoft.


87. Calvo, Angel (11 January 2018). "Top 10 most exciting reasons to migrate" (<https://techcommunity.microsoft.com/t5/PowerShell-AMA/Top-10-most-exciting-reasons-to-migrate/m-p/143960#M25>). *PowerShell AMA*. Microsoft.
88. Aiello, Joey (2018-09-13). "Announcing PowerShell Core 6.1" (<https://devblogs.microsoft.com/powershell/announcing-powershell-core-6-1/>). *devblogs.microsoft.com*. Microsoft. Retrieved 2019-06-01.
89. Lee, Steve (2019-03-28). "General Availability of PowerShell Core 6.2" (<https://devblogs.microsoft.com/powershell/general-availability-of-powershell-core-6-2/>). *devblogs.microsoft.com*. Microsoft. Retrieved 2019-06-01.
90. Mackie, Kurt (2019-05-30). "Microsoft Releases PowerShell 7 Preview" (<https://redmondmag.com/articles/2019/05/30/powershell-7-preview-released.aspx>). 1105 Media Inc. Retrieved 2019-06-01.
91. Lee, Steve (2019-04-05). "The Next Release of PowerShell – PowerShell 7" (<https://devblogs.microsoft.com/powershell/the-next-release-of-powershell-powershell-7/>). Microsoft. Retrieved 2019-06-01.
92. Lee, Steve (2019-05-30). "PowerShell 7 Road Map" (<https://devblogs.microsoft.com/powershell/powershell-7-road-map/>). *devblogs.microsoft.com*. Microsoft. Retrieved 2019-06-01.
93. "Test-Connection" (<https://technet.microsoft.com/en-us/library/hh849808.aspx>). *PowerShell documentations*. Microsoft. 9 August 2015.
94. "about_Script" (<https://technet.microsoft.com/en-us/library/hh847841.aspx>). *TechNet*. Microsoft. May 8, 2014.
95. "Import-LocalizedData" (<https://technet.microsoft.com/en-us/library/hh849919.aspx>). *TechNet*. Microsoft. May 8, 2014.
96. "about_Modules" (<https://technet.microsoft.com/en-us/library/hh847804.aspx>). *TechNet*. Microsoft. May 8, 2014.
97. "about_types.ps1xml" (<https://technet.microsoft.com/en-us/library/hh847881.aspx>). *TechNet*. Microsoft. May 8, 2014.
98. "Export-Clixml" (<https://technet.microsoft.com/en-us/library/hh849916.aspx>). *TechNet*. Microsoft. May 8, 2014.
99. "Export-Console" (<https://technet.microsoft.com/en-us/library/hh849706.aspx>). *TechNet*. Microsoft. May 8, 2014.
100. "about_Session_Configuration_Files" (<https://technet.microsoft.com/en-us/library/hh847838.aspx>). *TechNet*. Microsoft. May 8, 2014.
101. "Microsoft Transporter Suite for Lotus Domino" (<http://www.microsoft.com/downloads/details.aspx?familyid=35fc4205-792b-4306-8e4b-0de9cce72172&displaylang=en>). Retrieved 2008-03-07.
102. "PowerTools for Open XML" (<http://www.codeplex.com/PowerTools>). Retrieved 2008-06-20.
103. "MO74: WebSphere MQ – Windows PowerShell Library" (<http://www-1.ibm.com/support/docview.wss?rs=171&uid=swg24017698>). Retrieved 2007-12-05.
104. "PowerShell Commands for Active Directory by Quest Software" (<http://www.quest.com/powershell/activeroles-server.aspx>). Retrieved 2008-07-02.
105. "PowerShell Remoting through Group Policy" (<http://www.specopssoft.com/powershell/>). Retrieved 2007-12-07.
106. "VMware vSphere PowerCLI" (<https://www.vmware.com/go/PowerCLI>). Retrieved 2014-09-09.
107. "Windows PowerShell : IIS7 PowerShell Provider Tech Preview 2" (<http://blogs.msdn.com/powershell/archive/2008/07/03/iis7-powershell-provider-tech-preview-2.aspx>). Retrieved 2008-07-03.
108. "Kudos to the Win7 Diagnostics Team" (<http://blogs.msdn.com/powershell/archive/2009/06/14/kudos-to-the-win7-diagnostics-team.aspx>). Retrieved 2009-06-15.
109. Michael, Niehaus (10 Jul 2009). "MDT 2010 New Feature #16: PowerShell support" (<http://blogs.technet.com/b/mniehaus/archive/2009/07/10/mdt-2010-new-feature-16-powershell-support.aspx>). Retrieved 2014-10-27.
110. "Kudos to NetApp for Data ONTAP PowerShell Toolkit" (<http://blogs.msdn.com/b/powershell/archive/2010/06/16/kudos-to-netapp-for-data-ontap-powershell-toolkit.aspx>). Retrieved 2010-06-15.
111. "PowerShell Toolkit 4.2 Announcement" (<http://community.netapp.com/t5/Microsoft-Cloud-and-Virtualization-Discussions/NetApp-PowerShell-Toolkit-4-2-released/m-p/120539>). Retrieved 2016-09-07.
112. "Heterogeneous Job Scheduling With PowerShell" (<http://blogs.msdn.com/b/powershell/archive/2007/06/28/heterogeneous-job-scheduling-with-powershell.aspx>). Retrieved 2010-09-15.
113. "UIAutomation PowerShell Extensions" (<http://uiautomation.codeplex.com>). Retrieved 2012-02-16.
114. "EqualLogic HIT-ME with PowerShell" (<http://en.community.dell.com/techcenter/storage/w/wiki/2688.aspx>). Retrieved 2012-03-09.
115. de:LOGINventory

116. "Selenium PowerShell eXtensions" (<http://sepsx.codeplex.com>). Retrieved 2012-08-20.
117. "Pash" (<http://pash.sourceforge.net/>). *SourceForge*. Dice Holdings, Inc. Retrieved 2011-09-27.
118. "Pash Project" (<https://github.com/Pash-Project/Pash/>). Retrieved 2013-04-10.

Further reading

- Oakley, Andy (2005). *Monad (AKA PowerShell)*. O'Reilly Media. ISBN 0-596-10009-4.
- Holmes, Lee (2006). *Windows PowerShell Quick Reference*. O'Reilly Media. ISBN 0-596-52813-2.
- Holmes, Lee (2007). *Windows PowerShell Cookbook*. O'Reilly Media. ISBN 0-596-52849-3.
- Watt, Andrew (2007). *Professional Windows PowerShell*. Wrox Press. ISBN 0-471-94693-1.
- Kumaravel, Arul; White, Jon; Naixin Li, Michael; Happell, Scott; Xie, Guohui; Vutukuri, Krishna C. (2008). *Professional Windows PowerShell Programming: Snapins, Cmdlets, Hosts and Providers*. Wrox Press. ISBN 0-470-17393-9.
- Kopczynski, Tyson; Handley, Pete; Shaw, Marco (2009). *Windows PowerShell Unleashed* (2nd ed.). Pearson Education. ISBN 978-0-672-32988-3.
- Jones, Don; Hicks, Jeffery (2010). *Windows PowerShell 2.0: TFM* (3rd ed.). Sapien Technologies. ISBN 978-0-9821314-2-8.
- Finke, Douglas (2012). *Windows PowerShell for Developers*. O'Reilly Media. ISBN 1-4493-2270-0.
- Wilson, Ed (2013). *Windows PowerShell 3.0 Step by Step*. Microsoft Press. ISBN 978-0-7356-6339-8.
- Wilson, Ed (2014). *Windows PowerShell Best Practices*. Microsoft Press. ISBN 978-0-7356-6649-8.

External links

- [Official website \(https://microsoft.com/powershell\)](https://microsoft.com/powershell) 
- [PowerShell \(https://github.com/PowerShell/PowerShell\)](https://github.com/PowerShell/PowerShell) on GitHub
- *Windows PowerShell Survival Guide* (<http://social.technet.microsoft.com/wiki/contents/articles/183.windows-powershell-survival-guide-en-us.aspx>) on TechNet Wiki

Retrieved from "<https://en.wikipedia.org/w/index.php?title=PowerShell&oldid=914508235>"

This page was last edited on 7 September 2019, at 21:14 (UTC).

Text is available under the [Creative Commons Attribution-ShareAlike License](#); additional terms may apply. By using this site, you agree to the [Terms of Use](#) and [Privacy Policy](#). Wikipedia® is a registered trademark of the [Wikimedia Foundation, Inc.](#), a non-profit organization.