PowerShell

From Wikipedia, the free encyclopedia
Jump to navigation Jump to search

PowerShell

Screenshot of a PowerShell Core session in Windows Terminal

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; 15 years ago
Stable release	7.2.1 / December 15, 2021; 4 days ago[2]
Preview release	v7.2.0-rc.1[1] / October 23, 2021; 57 days ago
Typing discipline	Strong, safe, implicit and dynamic
Implementation language	C#
Platform	.NET Framework, .NET
OS	• Windows 7 and later • Windows Server 2008 R2 and later • macOS 10.12 and later • Ubuntu 14.04, 16.04, 18.04, and 20.04 • Debian 8.7+, 9, and 10 • CentOS 7 and 8 • Red Hat Enterprise Linux 7 and 8 • openSUSE 42.2, 42.3, 15.0, 15.1, 15.2 • Fedora 28, 29, 30
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) • .psrc (Role Capability File) • .cdxml (Cmdlet Definition XML Document)
Website	microsoft.com/powershell
Influenced by	
Python, Ksh, Perl, C#, CL, DCL, SQL, Tcl, Tk,[4] Chef, Puppet	

PowerShell or **Microsoft PowerShell** (formerly Windows PowerShell) is a task automation and configuration management program from Microsoft, consisting of a command-line shell and the 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 the .NET Framework, the latter on .NET Core. The name **Windows PowerShell** is still present on the latest versions of Windows 11 and 10, but the latest versions of PowerShell is called **PowerShell** or **Microsoft PowerShell**.

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, which may in turn be packaged into modules.

PowerShell provides 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. Updated local help contents can be retrieved from the Internet via the `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

- 1 Background
  - 1.1 Background
  - 1.2 Kermit
  - 1.3 Monad
  - 1.4 PowerShell
- 2 Design
  - 2.1 Grammar
  - 2.2 Named Commands
  - 2.3 Extended Type System
  - 2.4 Cmdlets
  - 2.5 Pipeline
  - 2.6 Scripting
  - 2.7 Hosting
- 3 Desired State Configuration
- 4 Versions
  - 4.1 Windows PowerShell 1.0
  - 4.2 Windows PowerShell 2.0
  - 4.3 Windows PowerShell 3.0
  - 4.4 Windows PowerShell 4.0
  - 4.5 Windows PowerShell 5.0
  - 4.6 Windows PowerShell 5.1
  - 4.7 PowerShell Core 6
  - 4.8 PowerShell 7
  - 4.9 PowerShell 7.2
- 5 Comparison of cmdlets with similar commands
- 6 Filename extensions
- 7 Application support
- 8 Alternative implementation
- 9 See also
- 10 References

- 11 Further reading
- 12 External links

Background[edit]

Background[edit]

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 the component object model (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.

Kermit[edit]

By the late 1990s, Intel had come to Microsoft asking for help in making Windows, which ran on Intel CPUs, a more appropriate platform to support the development of future Intel CPUs. At the time, Intel CPU development was accomplished on Sun Microsystems computers which ran Solaris (a Unix variant) on RISC-architecture CPUs. The ability to run Intel's many KornShell automation scripts on Windows was identified as a key capability. Internally, Microsoft began an effort to create a Windows port of Korn Shell, which was code-named Kermit.[11] Intel ultimately pivoted to a Linux-based development platform that could run on Intel CPUs, rendering the Kermit project redundant. However, with a fully funded team, Microsoft program manager Jeffrey Snover realized there was an opportunity to create a more general-purpose solution to Microsoft's problem of administrative automation.

Monad[edit]

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 called the "Monad Manifesto" by its chief architect, Jeffrey Snover.[12] In a 2017 interview, Snover explains the genesis of PowerShell, saying that he had been trying to make Unix tools available on Windows, which didn't work due to "core architectural difference[s] between Windows and Linux". Specifically, he noted that Linux considers everything an ASCII text file, whereas Windows considers everything an "API that returns structured data". They were fundamentally incompatible, which led him to take a different approach.[13]

Monad was to be a new extensible CLI with a fresh design capable of automating a range of core administrative tasks. Microsoft first demonstrated Monad publicly at the Professional Development Conference in Los Angeles in October 2003. A few months later, they opened up private beta, which

eventually led to a public beta. Microsoft published the first Monad public beta release on 17 June 2005 and the Beta 2 on 11 September 2005, and Beta 3 on 10 January 2006.

PowerShell[edit]

On 25 April 2006, not long after the initial Monad announcement, Microsoft announced that Monad had been renamed **Windows PowerShell**, positioning it as a significant part of its management technology offerings.[14] Release Candidate (RC) 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, rather than a mere add-on.

Release Candidate 2 of PowerShell version 1 was released on 26 September 2006, with final release to the web on 14 November 2006. PowerShell for earlier versions of Windows was released on 30 January 2007.[15] 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 for Linux 6.0 Alpha 9
on Ubuntu 14.04 x64

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.[16] In an October 2009 issue of *TechNet Magazine*, Microsoft called proficiency with PowerShell "the single most important skill a Windows administrator will need in the coming years".[17]

Windows 10 shipped a testing framework for PowerShell.[18]

On 18 August 2016, Microsoft announced[19] 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.[20] 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.[21] Starting with version 5.1, PowerShell Core is bundled with Windows Server 2016 Nano Server.[22][23]

Design[edit]

A key design tactic for PowerShell was to leverage the large number of APIs that already existed in Windows, Windows Management Instrumentation, .NET Framework, and other software. PowerShell cmdlets “wrap around” existing functionality. The intent with this tactic is to provide an administrator-friendly, more-consistent interface between administrators and a wide range of underlying functionality. With PowerShell, an administrator doesn't need to know .NET, WMI, or low-level API coding, and can instead focus on using the cmdlets exposed by PowerShell. In this regard, PowerShell creates little new functionality, instead focusing on making existing functionality more accessible to a particular audience.[24]

Grammar[edit]

PowerShell's developers based the core grammar of the tool on that of the POSIX 1003.2 KornShell.[25]

However, PowerShell's language was also influenced by PHP, Perl, and many other existing languages.[26]

Named Commands[edit]

Windows PowerShell can execute four kinds of named commands:[27]

- *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, where the commands can be entered and their output displayed. The user interface 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.[*citation needed*]

Extended Type System[edit]

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.[28]

Cmdlets[edit]

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*, which makes it self-documenting code.[29] 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.[29]

Cmdlets are specialized .NET classes, which the PowerShell runtime instantiates and invokes at execution time. Cmdlets derive either from `Cmdlet` or from `PSCmdlet`, the latter being used when the cmdlet needs to interact with the PowerShell runtime.[29] 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.[30] 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.[31][32]

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`[33] 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.[34]

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.[35] 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 a database), they can add specific providers as well.[*citation needed*]

The number of cmdlets included in the base PowerShell install has generally increased with each version:

Version	Cmdlets
Windows PowerShell 1.0	129[36]
Windows PowerShell 2.0	632[37]
Windows PowerShell 3.0	about 1,000[38]
Windows PowerShell 4.0	?
Windows PowerShell 5.0	about 1,300[39]
Windows PowerShell 5.1	1586[*citation needed*]
PowerShell Core 6.0	?
PowerShell Core 6.1	?
PowerShell Core 6.2	?
PowerShell 7.0	1507[*citation needed*]
PowerShell 7.1	?

Cmdlets can be added into the shell through snap-ins (deprecated in v2) and modules; users are not limited to the cmdlets included in the base PowerShell installation.

Pipeline[edit]

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 ten processes based on handle count.[*citation needed*]

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. Additionally, 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.[40] An object can also encapsulate certain functions that work on the contained data, which become available to the recipient command for use.[41][42] 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.[43][44]

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 backward 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.[45][46][47]

Scripting[edit]

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,[48] 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.[*citation needed*]

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.[49] 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.[50][51]

Using the `function` keyword, PowerShell provides for the creation of functions. A simple function has the following general look:[52]

```
function name ([Type]$Param1, [Type]$Param2)
{
  # Instructions
}
```

However, PowerShell allows for advanced functions that support named parameters, positional parameters, switch parameters and dynamic parameters.[52]

```
function Verb-Noun
{
  param (
    # Definition of static parameters
  )
  dynamicparam {
    # Definition of dynamic parameters
  }
  begin {
    # Set of instruction to run at the start of the pipeline
  }
  process {
    # Main instruction sets, ran for each item in the pipeline
  }
  end {
    # Set of instruction to run at the end of the pipeline
  }
}
```

The defined function is invoked in either of the following forms:[52]

```
name value1 value2
Verb-Noun -Param1 value1 -Param2 value2
```

PowerShell allows any static .NET methods to be called by providing their namespaces enclosed in brackets (`[]`), and then using a pair of colons (`::`) to indicate the static method.[53] For example:

```
[Console]::WriteLine("PowerShell")
```

There are dozens of ways to create objects in PowerShell. Once created, one can access the properties and instance methods of an object using the `.` notation.[53]

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.[54]

The following list of special characters is supported by PowerShell:[55]

PowerShell special characters

Sequence	Meaning
`0	Null
`a	Alert
`b	Backspace
`e	Escape
`f	Form feed
`n	Newline
`r	Carriage return
`t	Horizontal tab
`u{x}	Unicode escape sequence
`v	Vertical tab
--%	Treat any character from this point forward literally

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.[56]

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.[57] 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.[58]

The PowerShell scripting language supports binary prefix notation similar to the scientific notation supported by many programming languages in the C-family.[59]

Hosting[edit]

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.[*citation needed*]

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.[*citation needed*]

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.[*citation needed*]

Desired State Configuration[edit]

DSC allows for declaratively specifying how a software environment should be configured.[60]

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[edit]

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

Windows PowerShell 1.0[edit]

Windows PowerShell 1.0 session
using the Windows Console

PowerShell 1.0 was released in November 2006 for Windows XP SP2, Windows Server 2003 SP1 and Windows Vista.[61] It is an optional component of Windows Server 2008.

Windows PowerShell 2.0[edit]

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

PowerShell 2.0 is integrated with Windows 7 and Windows Server 2008 R2[62] and is released for Windows XP with Service Pack 3, Windows Server 2003 with Service Pack 2, and Windows Vista with Service Pack 1.[63][64]

PowerShell v2 includes changes to the scripting language and hosting API, in addition to including more than 240 new cmdlets.[65][66]

New features of PowerShell 2.0 include:[67][68][69]

- **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".[70]
- **SteppablePipelines**: 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).[71]
- **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.[72]
- **Block comments**: PowerShell 2.0 supports block comments using `<#` and `#>` as delimiters.[73]
- **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 transaction managed by PowerShell

Windows PowerShell 3.0[edit]

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.[74][75]

PowerShell 3.0 is part of a larger package, Windows Management Framework 3.0 (WMF3), which also

contains the WinRM service to support remoting.[75] 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.[76] Windows Management Framework 3.0 was released for general availability in December 2012[77] and is included with Windows 8 and Windows Server 2012 by default.[78]

New features in PowerShell 3.0 include:[75][79]: 33–34

- **Scheduled jobs**: Jobs can be scheduled to run on a preset time and date using the Windows Task Scheduler infrastructure.
- **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 printers, which had previously been performed via WMI.[*further explanation needed*]

Windows PowerShell 4.0[edit]

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.[80]

New features in PowerShell 4.0 include:

- **Desired State Configuration**:[81][82][83] 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.

Windows PowerShell 5.0[edit]

PowerShell 5.0 icon

Windows Management Framework (WMF) 5.0 RTM which includes PowerShell 5.0 was re-released to web on 24 February 2016, following an initial release with a severe bug.[84]

Key features included:

- The new `class` keyword that creates classes for object-oriented programming

- The new `enum` keyword that creates enums
- `OneGet` cmdlets to support the Chocolatey package manager[85]
- Extending support for switch management to layer 2 network switches.[86]
- Debugging for PowerShell background jobs and instances of PowerShell hosted in other processes (each of which is called a "runspace")
- 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

Windows PowerShell 5.1[edit]

It was released along with the Windows 10 Anniversary Update[87] on August 2, 2016, and in Windows Server 2016.[88] 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.[89] 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,[90] and was released on January 19, 2017.[91]

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 the .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.[23] This was the final version of PowerShell made exclusively for Windows.

PowerShell Core 6[edit]

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.[92] 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.[93]

The most significant change in this version of PowerShell is the expansion to the other platforms. For Windows administrators, this version of PowerShell did not include 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.[94]

According to Microsoft, one of the new features of PowerShell 6.1 is "Compatibility with 1900+ existing cmdlets in Windows 10 and Windows Server 2019."[95] Still, no details of these cmdlets can be found in the full version of the change log.[96] Microsoft later professes that this number was insufficient as PowerShell Core failed to replace Windows PowerShell 5.1 and gain traction on Windows.[97] It was, however, popular on Linux.[97]

PowerShell Core 6.2 is focused primarily on performance improvements, bug fixes, and smaller cmdlet and language enhancements that improved developer productivity.[98]

PowerShell 7[edit]

PowerShell 7 is the replacement for PowerShell Core 6.x products as well as Windows PowerShell 5.1,

which is the last supported Windows PowerShell version.[99][97] The focus in development was to make PowerShell 7 a viable replacement for Windows PowerShell 5.1, i.e. to have near parity with Windows PowerShell in terms of compatibility with modules that ship with Windows.[100]

New features in PowerShell 7 include:[101]

- The `-Parallel` switch for the `ForEach-Object` cmdlet to help handle parallel processing
- Near parity with Windows PowerShell in terms of compatibility with built-in Windows modules
- A new error view
- The `Get-Error` cmdlet
- Pipeline chaining operators (`&&` and `||`) that allow conditional execution of the next cmdlet in the pipeline
- The ?: operator for ternary operation
- The `??=` operator that only assigns a value to a variable when the variable's existing value is null
- The `??` operator for null coalescing
- Cross-platform `Invoke-DscResource` (experimental)
- Return of the `Out-GridView` cmdlet
- Return of the `-ShowWindow` switch for the `Get-Help`

PowerShell 7.2[edit]

PowerShell 7.2 was released together with .NET 6.0 and is available for Linux, Windows, Mac, and in a Docker container format.[102]

New features are

- universal installer packages for Linux
- support for Windows Microsoft Update
- improved tab completions
- PSReadLine 2.1 with predictive IntelliSense

Comparison of cmdlets with similar commands[edit]

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	dir	ls	Lists all files and folders in the current or given folder
Test-Connection[a]	ping	ping	ping	Sends ICMP echo requests to the specified machine from the current machine, or instructs another machine to do so
Get-Content	gc, type, cat	type	cat	Gets the content of a file
Get-Command	gcm	help, where	type, which, compgen	Lists available commands and gets command path
Get-Help	help, man	help	apropos, man	Prints a command's documentation on the console
Clear-Host	cls, clear	cls	clear	Clears the screen[b]
Copy-Item	cpi, copy, cp	copy, xcopy, robocopy	cp	Copies files and folders to another location
Move-Item	mi, move, mv	move	mv	Moves files and folders to a new location

Remove-Item	ri, del, erase, rmdir, rd, rm	del, erase, rmdir, rd	rm, rmdir	Deletes files or folders
Rename-Item	rni, ren, mv	ren, rename	mv	Renames a single file, folder, hard link or symbolic link
Get-Location	gl, cd, pwd	cd	pwd	Displays the working path (current folder)
Pop-Location	popd	popd	popd	Changes the working path to the location most recently pushed onto the stack
Push-Location	pushd	pushd	pushd	Stores the working path onto the stack
Set-Location	sl, cd, chdir	cd, chdir	cd	Changes the working path
Tee-Object	tee	N/A	tee	Pipes input to a file or variable, passing the input along the pipeline
Write-Output	echo, write	echo	echo	Prints strings or other objects to the standard output
Get-Process	gps, ps	tlist,[c] tasklist[d]	ps	Lists all running processes
Stop-Process	spps, kill	kill,[c] taskkill[d]	kill[e]	Stops a running process
Select-String	sls	findstr	find, grep	Prints lines matching a pattern
Set-Variable	sv, set	set	env, export, set, setenv	Creates or alters the contents of an environment variable
Invoke-WebRequest	iwr, ~~curl, wget~~[f]	curl[104]	wget, 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.[103]
b. ^ Clear-Host is implemented as a predefined PowerShell function.
c. ^ ***a*** ***b*** Available in Windows NT 4, Windows 98 Resource Kit, Windows 2000 Support Tools
d. ^ ***a*** ***b*** Introduced in Windows XP Professional Edition
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.

Filename extensions[edit]

Extension	Description
.ps1	Script file[105]
.psd1	Module's manifest file; usually comes with a script module or binary module[106]
.psm1	Script module file[107]
.dll	DLL-compliant[a] binary module file[108]
.ps1xml	Format and type definitions file[47][109]
.xml	XML-compliant[b] serialized data file[110]
.psc1	Console file[111]
.pssc	Session configuration file[112]
.psrc	Role Capability file[113]

a. ^ Dynamic-link library (DLL) is not a PowerShell-only format. It is a generic format for storing compiled .NET assembly's code.
b. ^ XML is not a PowerShell-only format. It is a popular information interchange format.

Application support[edit]

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[114]	08.02.0012	47	No	No
Microsoft PowerTools for Open XML[115]	1.0	33	No	No
IBM WebSphere MQ[116]	6.0.2.2	44	No	No
IoT Core Add-ons[117]		74	Unknown	Unknown
Quest Management Shell for Active Directory[118]	1.7	95	No	No
Special Operations Software Specops Command[119]	1.0	Yes	No	Yes
VMware vSphere PowerCLI[120]	6.5 R1	500+	Yes	Yes
Internet Information Services[121]	7.0	54	Yes	No
Windows 7 Troubleshooting Center[122]	6.1	Yes	No	Yes
Microsoft Deployment Toolkit[123]	2010	Yes	Yes	Yes
NetApp PowerShell Toolkit[124][125]	4.2	2000+	Yes	Yes
JAMS Scheduler – Job Access & Management System[126]	5.0	52	Yes	Yes
UIAutomation[127]	0.8	432	No	No
Dell Equallogic[128]	3.5	55	No	No
LOGINventory[129]	5.8	Yes	Yes	Yes
SePSX[130]	0.4.1	39	No	No

Alternative implementation[edit]

A project named *Pash*, a pun on the widely known "bash" Unix shell, has been an open-source and cross-platform reimplementation of PowerShell via the Mono framework.[131] Pash was created by Igor Moochnick, written in C# and was released under the GNU General Public License. Pash development stalled in 2008, was restarted on GitHub in 2012,[132] and finally ceased in 2016 when PowerShell was officially made open-source and cross-platform.[133]

See also[edit]

- Common Information Model (computing)
- Comparison of command shells
- Comparison of programming languages
- Web-Based Enterprise Management
- Windows Script Host
- Windows Terminal

References[edit]

1. ^ "PowerShell 7.2.0 RC1 now available for testing - MSPoweruser".
2. ^ "PowerShell/PowerShell". *GitHub*. Retrieved 2021-12-15.
3. ^ "PowerShell for every system!". 12 June 2017 – via GitHub.
4. ^ Snover, Jeffrey (May 25, 2008). "PowerShell and WPF: WTF". *Windows PowerShell Blog*. Microsoft.

5. ^ **a** **b** **c** Bright, Peter (2016-08-18). "PowerShell is Microsoft's latest open source release, coming to Linux, OS X". *Ars Technica*. Condé Nast. Archived from the original on 2020-04-09. Retrieved 2020-05-12.
6. ^ **a** **b** **c** **d** **e** "How Windows PowerShell works". *Microsoft Developer Network*. Microsoft. Retrieved 2007-11-27.
7. ^ Truher, Jim (December 2007). "Extend Windows PowerShell With Custom Commands". *MSDN Magazine*. Microsoft. Archived from the original on 6 October 2008.
8. ^ Lowe, Scott (January 4, 2007). "Exchange 2007: Get used to the command line". *TechRepublic*. CBS Interactive. Archived from the original on 2018-11-16. Retrieved 2020-05-12.
9. ^ Snover, Jeffrey (2007-11-13). "SQL Server Support for PowerShell!". *Windows PowerShell Blog* (blog posting). Microsoft. Archived from the original on 2007-11-15. Retrieved 2007-11-13.
10. ^ Dragan, Richard V. (April 23, 2003). "Windows Server 2003 Delivers Improvements All Around". 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. ^ Jones, Don (2020). *Shell of an Idea: The Untold History of PowerShell*. p. 25. ISBN 978-1-9536450-3-6.
12. ^ Jeffrey P. Snover (8 August 2002). "Monad Manifesto" (PDF). *Windows PowerShell Blog*. Microsoft. Retrieved 2 April 2021.
13. ^ Biggar and Harbaugh (2017-09-14). "The Man Behind Windows PowerShell". *To Be Continuous* (Podcast). Heavybit. Retrieved 2017-09-14.
14. ^ "Windows PowerShell (Monad) Has Arrived". *Windows PowerShell Blog*. Microsoft. April 25, 2006.
15. ^ Snover, Jeffrey (November 15, 2006). "Windows PowerShell & Windows Vista". *Windows PowerShell Blog* (blog posting). Microsoft.
16. ^ "Windows Management Framework (Windows PowerShell 2.0, WinRM 2.0, and BITS 4.0)". *Support*. Microsoft. September 30, 2013. Archived from the original on October 13, 2013.
17. ^ Posey, Brien (6 October 2009). "10 reasons why you should learn to use PowerShell". *TechRepublic*. Retrieved 2 April 2021.
18. ^ "What is Pester and Why Should I Care?". 14 December 2015.
19. ^ Snover, Jeffrey (18 August 2016). "PowerShell is open sourced and is available on Linux". *Microsoft Azure Blog*. Microsoft.
20. ^ "PowerShell/PowerShell". *GitHub*. Retrieved 2016-08-18.
21. ^ Hansen, Kenneth; Calvo, Angel (August 18, 2016). "PowerShell on Linux and Open Source!". *Windows PowerShell Blog*. Microsoft.
22. ^ Foley, Mary Jo (August 18, 2016). "Microsoft open sources PowerShell; brings it to Linux and Mac OS X". *ZDNet*. CBS Interactive.
23. ^ **a** **b** "PowerShell on Nano Server". *TechNet*. Microsoft. 20 October 2016.
24. ^ Jones, Don (2020). *Shell of an Idea: The Untold History of PowerShell*. p. 45. ISBN 978-1-9536450-3-6.
25. ^ Payette, Bruce (2007). *Windows PowerShell in Action*. 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."
26. ^ Jones, Don (2020). *Shell of an Idea: The Untold History of PowerShell*. p. 109. ISBN 978-1-9536450-3-6.
27. ^ "about_Command_Precedence". *TechNet*. Microsoft. May 8, 2014.
28. ^ "Windows PowerShell Extended Type System". Retrieved 2007-11-28.
29. ^ **a** **b** **c** "Windows PowerShell Cmdlets". Retrieved 2007-11-28.
30. ^ "Creating Your First Cmdlet". Retrieved 2007-11-28.
31. ^ "Get-Verb". *TechNet*. Microsoft. May 8, 2014.
32. ^ "Cmdlet Overview". *MSDN*. Microsoft. May 8, 2014.
33. ^ "Adding parameters That Process Command Line Input". Retrieved 2007-11-28.
34. ^ "Adding parameters That Process Pipeline Input". Retrieved 2007-11-28.
35. ^ "Windows PowerShell Providers". Retrieved 2010-10-14.
36. ^ Yoshizawa, Tomoaki; Ramos, Durval (29 September 2012). "PowerShell 1.0 Cmdlets". *TechNet Articles*. Microsoft.
37. ^ Yoshizawa, Tomoaki (10 July 2012). "PowerShell 2.0 Cmdlets". *TechNet Articles*. Microsoft.
38. ^ Wilson, Ed (2013). "1: Overview of Windows PowerShell 3.0". *Windows Powershell 3.0 Step by Step*. Sebastopol, California: Microsoft Press. ISBN 978-0-7356-7000-6. OCLC 829236530. "Windows PowerShell 3.0 comes with about 1,000 cmdlets on Windows 8"
39. ^ Wilson, Ed (2015). "1: Overview of Windows PowerShell 5.0". *Windows PowerShell Step by Step* (Third ed.). Redmond, Washington: Microsoft Press. ISBN 978-1-5093-0043-3. OCLC 927112976. "Windows PowerShell 5.0 comes with about 1,300 cmdlets on Windows 10"
40. ^ "Windows PowerShell Owner's Manual: Piping and the Pipeline in Windows PowerShell". *TechNet*. Microsoft. Retrieved 2011-09-27.
41. ^ Jones, Don (2008). "Windows PowerShell – Rethinking the Pipeline". *Microsoft TechNet*. Microsoft. Retrieved 2007-11-28.

42. ^ "Windows PowerShell Object Concepts". Archived from the original on August 19, 2007. Retrieved 2007-11-28.
43. ^ "How PowerShell Formatting and Outputting REALLY works". Retrieved 2007-11-28.
44. ^ "More – How does PowerShell formatting really work?". Retrieved 2007-11-28.
45. ^ "about_Pipelines". *TechNet*. Microsoft. May 8, 2014.
46. ^ "about_Objects". *TechNet*. Microsoft. May 8, 2014.
47. ^ **a** **b** "about_Format.ps1xml". *TechNet*. Microsoft. May 8, 2014.
48. ^ "Anonymous Functions and Code Blocks in PowerShell". Retrieved 2012-01-21.
49. ^ "Introduction to Windows PowerShell's Variables". Retrieved 2007-11-28.
50. ^ "Byte Conversion". *Windows PowerShell Tip of the Week*. Retrieved 15 November 2013.
51. ^ Ravikanth (20 May 2013). "Converting to size units (KB, MB, GB, TB, and PB) without using PowerShell multipliers". *PowerShell Magazine*.
52. ^ **a** **b** **c** "about_Functions". *Microsoft TechNet*. Microsoft. 17 October 2013. Retrieved 15 November 2013.
53. ^ **a** **b** "Lightweight Testing with Windows PowerShell". Retrieved 2007-11-28.
54. ^ Angelopoulos, Alex; Karen, Bemowski (4 December 2007). "PowerShell Got Smart About Smart Quotes". *Windows IT Pro*. Penton Media. Retrieved 15 November 2013.
55. ^ "About Special Characters". Powershell / Scripting. *Microsoft*. June 8, 2017. Retrieved June 20, 2019.
56. ^ "Trap [Exception] { "In PowerShell" }". Retrieved 2007-11-28.
57. ^ "Running Windows PowerShell Scripts". *Microsoft*. Retrieved 2007-11-28.
58. ^ "about_Signing". *Microsoft TechNet*. Microsoft. 17 October 2013. Retrieved 15 November 2013.
59. ^ Lee Holmes (September 2006). *Windows PowerShell Quick Reference*. O'Reilly Media.
60. ^ eslesar. "Windows PowerShell Desired State Configuration Overview". *msdn.microsoft.com*.
61. ^ Chung, Leonard; Snover, Jeffrey; Kumaravel, Arul (14 November 2006). "It's a Wrap! Windows PowerShell 1.0 Released!". *Windows PowerShell Blog*. Microsoft.
62. ^ "PowerShell will be installed by default on Windows Server 08 R2 (WS08R2) and Windows 7 (W7)!". *Windows PowerShell Blog*. Microsoft. 2008-10-28. Retrieved 2011-09-27.
63. ^ "Windows Management Framework is here!". 2009-10-27. Retrieved 2009-10-30.
64. ^ "Microsoft Support Knowledge Base: Windows Management Framework (Windows PowerShell 2.0, WinRM 2.0, and BITS 4.0)". Support.microsoft.com. 2011-09-23. Retrieved 2011-09-27.
65. ^ "574 Reasons Why We Are So Proud and Optimistic About W7 and WS08R2". *Windows PowerShell Blog*. Microsoft. 2008-10-29. Retrieved 2011-09-27.
66. ^ Snover, Jeffrey (2008). "PowerShell: Creating Manageable Web Services". Archived from the original on October 13, 2008. Retrieved July 19, 2015.
67. ^ "What's New in CTP of PowerShell 2.0". Retrieved 2007-11-28.
68. ^ "Windows PowerShell V2 Community Technology Preview 2 (CTP2) – releaseNotes". *Microsoft*. Archived from the original on May 6, 2008. Retrieved 2008-05-05.
69. ^ "Differences between PowerShell 1.0 and PowerShell 2.0". Retrieved 2010-06-26.
70. ^ Jones, Don (May 2010). "Windows PowerShell: Writing Cmdlets in Script". *TechNet Magazine*. Microsoft.
71. ^ "GoGrid Snap-in – Managing Cloud Services with PowerShell". *Windows PowerShell Blog*. Microsoft. 2008-10-14. Retrieved 2011-09-27.
72. ^ "Emit-XML". *Windows PowerShell Blog*. Microsoft. 2008-10-17. Retrieved 2011-09-27.
73. ^ "Block Comments in V2". *Windows PowerShell Blog*. Microsoft. 2008-06-14. Retrieved 2011-09-27.
74. ^ Lee, Thomas (13 August 2012). "PowerShell Version 3 is RTM!". *Under The Stairs*. Retrieved 2012-08-13.
75. ^ **a** **b** **c** "Windows Management Framework 3.0". *Download Center*. Microsoft. 4 September 2012. Retrieved 2012-11-08.
76. ^ "Windows Management Framework 3.0 Community Technology Preview (CTP) #2 Available for Download". *Windows PowerShell Blog*. Microsoft. 2 December 2011.
77. ^ "Windows Management Framework 3.0". *Download Center*. Microsoft. 3 December 2012.
78. ^ Jofre, JuanPablo (December 14, 2016). "Windows PowerShell System Requirements". *Microsoft Developer Network*. Microsoft. Retrieved April 20, 2017.
79. ^ Honeycutt, Jerry (2012). Woolley, Valerie (ed.). *Introducing Windows 8: An Overview for IT Professionals*. Redmond, WA: Microsoft Press. ISBN 978-0-7356-7050-1.
80. ^ "Windows Management Framework 4.0 is now available". Microsoft. 24 October 2013. Retrieved 4 November 2013.
81. ^ Levy, Shay (25 June 2013). "New Features in Windows PowerShell 4.0". *PowerShell Magazine*. Retrieved 26 June 2013.
82. ^ "Desired State Configuration in Windows Server 2012 R2 PowerShell". *Channel 9*. Microsoft. 3 June 2013. Retrieved 26 June 2013.
83. ^ Hall, Adrian (7 June 2013). "Thoughts from Microsoft TechEd North America". *Blogs: Tips & Tricks*. Splunk. Retrieved 26 June 2013.

84. ^ "Windows Management Framework (WMF) 5.0 RTM packages has been republished". *Windows PowerShell Blog*. Microsoft. February 24, 2016.
85. ^ "Q and A". *GitHub*. Retrieved 21 April 2015.
86. ^ Snover, Jeffrey (2014-04-03). "Windows Management Framework V5 Preview". *blogs.technet.com*. Microsoft. Archived from the original on 2014-06-30. Retrieved 2015-04-21.
87. ^ says, Jaap Brasser (2 August 2016). "#PSTip New PowerShell Commands in Windows 10 Anniversary Update".
88. ^ "What's New In Windows Server 2016 Standard Edition Part 9 – Management And Automation".
89. ^ "Microsoft.PowerShell.LocalAccounts Module". *technet.microsoft.com*.
90. ^ "Announcing Windows Management Framework (WMF) 5.1 Preview". 16 July 2016.
91. ^ "WMF 5.1". *Microsoft Download Center*.
92. ^ Aiello, Joey (11 January 2018). "PowerShell Core 6.0: Generally Available (GA) and Supported!". *PowerShell Team Blog*. Microsoft. Archived from the original on 11 June 2018. Retrieved 11 June 2018.
93. ^ Aiello, Joey; Wheeler, Sean (10 January 2018). "PowerShell Core Support Lifecycle". *Microsoft Docs*. Microsoft.
94. ^ Calvo, Angel (11 January 2018). "Top 10 most exciting reasons to migrate". *PowerShell AMA*. Microsoft.
95. ^ Aiello, Joey (2018-09-13). "Announcing PowerShell Core 6.1". *devblogs.microsoft.com*. Microsoft. Retrieved 2019-06-01.
96. ^ "PowerShell/PowerShell". *GitHub*. Retrieved 2020-06-22.
97. ^ ***a*** ***b*** ***c*** Lee, Steve (2019-04-05). "The Next Release of PowerShell – PowerShell 7". Microsoft. Retrieved 2019-06-01.
98. ^ Lee, Steve (2019-03-28). "General Availability of PowerShell Core 6.2". *devblogs.microsoft.com*. Microsoft. Retrieved 2019-06-01.
99. ^ Mackie, Kurt (2019-05-30). "Microsoft Releases PowerShell 7 Preview". 1105 Media Inc. Retrieved 2019-06-01.
100. ^ Lee, Steve (2019-05-30). "PowerShell 7 Road Map". *devblogs.microsoft.com*. Microsoft. Retrieved 2020-08-12.
101. ^ PowerShell 7 Preview 5 | PowerShell
102. ^ "PowerShell 7.2 is the new version of Microsoft's next-generation shell - itsfoss.net". 12 November 2021.
103. ^ "Test-Connection". *PowerShell documentations*. Microsoft. 9 August 2015.
104. ^ Tar and Curl Come to Windows! - Microsoft Tech Community - 382409
105. ^ Wheeler, Sean (2 June 2020). "About Scripts". *Microsoft Docs*. Microsoft.
106. ^ Wheeler, Sean; Smatlak, David; Wilson, Chase (16 October 2019). "How to write a PowerShell module manifest". *Docs*. Microsoft.
107. ^ Wheeler, Sean; Smatlak, David (22 November 2019). "How to Write a PowerShell Script Module". *Microsoft Docs*. Microsoft.
108. ^ Wheeler, Sean (13 November 2016). "How to Write a PowerShell Binary Module". *Microsoft Docs*. Microsoft.
109. ^ Wheeler, Sean; Jofre, Juan Pablo; Vorobev, Sergei; Nikolaev, Kirill; Coulter, David (2 June 2020). "About Types.ps1xml". *Microsoft Docs*. Microsoft.
110. ^ Wheeler, Sean. "Export-Clixml". *Microsoft Docs*. Microsoft.
111. ^ Wheeler, Sean; Jofre, Juan Pablo; Vorobev, Sergei; Nikolaev, Kirill; Coulter, David. "Export-Console". *Microsoft Docs*. Microsoft.
112. ^ Wheeler, Sean (2 June 2020). "About Session Configuration Files". *Microsoft Docs*. Microsoft.
113. ^ Wheeler, Sean (2 June 2020). "New-PSRoleCapabilityFile". *Microsoft Docs*. Microsoft.
114. ^ "Microsoft Transporter Suite for Lotus Domino". *Microsoft*. Retrieved 2008-03-07.
115. ^ "PowerTools for Open XML". Retrieved 2008-06-20.
116. ^ "MO74: WebSphere MQ – Windows PowerShell Library". Retrieved 2007-12-05.
117. ^ "IoT Core Add-ons command-line options". Retrieved 2020-06-13.
118. ^ "PowerShell Commands for Active Directory by Quest Software". Retrieved 2008-07-02.
119. ^ "PowerShell Remoting through Group Policy". Retrieved 2007-12-07.
120. ^ "VMware vSphere PowerCLI". Retrieved 2014-09-09.
121. ^ "Windows PowerShell : IIS7 PowerShell Provider Tech Preview 2". Retrieved 2008-07-03.
122. ^ "Kudos to the Win7 Diagnostics Team". Retrieved 2009-06-15.
123. ^ Michael, Niehaus (10 Jul 2009). "MDT 2010 New Feature #16: PowerShell support". Retrieved 2014-10-27.
124. ^ "Kudos to NetApp for Data ONTAP PowerShell ToolKit". Retrieved 2010-06-15.
125. ^ "PowerShell Toolkit 4.2 Announcement". Retrieved 2016-09-07.
126. ^ "Heterogeneous Job Scheduling With PowerShell". Retrieved 2010-09-15.
127. ^ "UIAutomation PowerShell Extensions". Retrieved 2012-02-16.
128. ^ "EqualLogic HIT-ME with PowerShell". Retrieved 2012-03-09.
129. ^ de:LOGINventory
130. ^ "Selenium PowerShell eXtensions". Retrieved 2012-08-20.

131. ^ "Pash". *SourceForge*. Dice Holdings, Inc. Retrieved 2011-09-27.
132. ^ "Pash Project". *GitHub*. Retrieved 2013-04-10.
133. ^ "Pash is now obsolete · Issue #429 · Pash-Project/Pash". *GitHub*. Retrieved 2019-11-26.

Further reading[edit]

- Finke, Douglas (2012). *Windows PowerShell for Developers*. O'Reilly Media. ISBN 978-1-4493-2270-0.
- 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 978-0-596-52849-2.
- Jones, Don; Hicks, Jeffery (2010). *Windows PowerShell 2.0: TFM* (3rd ed.). Sapien Technologies. ISBN 978-0-9821314-2-8.
- Jones, Don (2020). *Shell of an Idea: The Untold History of PowerShell*. Self-published. ISBN 978-1-9536450-3-6.
- Kopczynski, Tyson; Handley, Pete; Shaw, Marco (2009). *Windows PowerShell Unleashed* (2nd ed.). Pearson Education. ISBN 978-0-672-32988-3.
- 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 978-0-470-17393-0.
- Oakley, Andy (2005). *Monad (AKA PowerShell)*. O'Reilly Media. ISBN 0-596-10009-4.
- Watt, Andrew (2007). *Professional Windows PowerShell*. Wrox Press. ISBN 978-0-471-94693-9.
- 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[edit]

Wikiversity has learning resources about ***PowerShell***

- Official website
- PowerShell on GitHub
- PowerShellExplained.com
- *Windows PowerShell Survival Guide* on TechNet Wiki

Microsoft Windows components

- v
- t
- e

Management tools	App Installer Command Prompt Control Panel Applets Device Manager Disk Cleanup Disk Defragmenter Driver Verifier DxDiag Event Viewer IExpress

	Management Console Netsh Performance Monitor Recovery Console Resource Monitor Settings Sysprep System Configuration System File Checker System Information System Policy Editor System Restore Task Manager Windows Error Reporting Windows Ink Windows Installer PowerShell Windows Update Windows Insider WinRE WMI
Apps	3D Viewer Alarms & Clock Calculator Calendar Camera Character Map Cortana Edge Fax and Scan Feedback Hub Get Help Groove Music Magnifier Mail Maps Messaging Movies & TV Mobility Center Money News Narrator Notepad OneDrive

	OneNote Paint Paint 3D People Photos Quick Assist Snip & Sketch Speech Recognition Skype Sports Sticky Notes Store Tips Voice Recorder Weather WordPad Xbox Console Companion Your Phone
Shell	Action Center Aero AutoPlay AutoRun ClearType Explorer Search Indexing Service IFilter Saved search Namespace Special folder Start menu Taskbar Task View Windows Spotlight Windows XP visual styles
Services	Service Control Manager BITS CLFS Multimedia Class Scheduler Shadow Copy Task Scheduler Error Reporting

	Wireless Zero Configuration
File systems	CDFS DFS exFAT IFS FAT NTFS Hard link links Mount Point Reparse point TxF EFS ReFS UDF
Server	Domains Active Directory DNS Group Policy Roaming user profiles Folder redirection Distributed Transaction Coordinator MSMQ Windows Media Services Active DRM Services IIS WSUS SharePoint Network Access Protection PWS DFS Replication Print Services for UNIX Remote Desktop Services Remote Differential Compression Remote Installation Services Windows Deployment Services System Resource Manager Hyper-V Server Core
Architecture	Architecture of Windows NT

	Startup process NT NT 6 CSRSS Desktop Window Manager Portable Executable EXE DLL Enhanced Write Filter Graphics Device Interface hal.dll I/O request packet Imaging Format Kernel Transaction Manager Library files Logical Disk Manager LSASS MinWin NTLDR Ntoskrnl.exe Object Manager Open XML Paper Specification Registry Resource Protection Security Account Manager Server Message Block Shadow Copy SMSS System Idle Process USER WHEA Win32 console Winlogon WinUSB
Security	Security and Maintenance AppLocker BitLocker Credential Guard Data Execution Prevention Family Safety Kernel Patch Protection Mandatory Integrity Control Protected Media Path

		User Account Control User Interface Privilege Isolation Windows Defender Windows Firewall
Compatibility		COMMAND.COM WoW64 Windows Subsystem for Linux
API		Active Scripting WSH VBScript JScript COM ActiveX ActiveX Document COM Structured storage DCOM OLE OLE Automation Transaction Server DirectX .NET Framework Universal Windows Platform Windows Mixed Reality Windows Runtime WinUSB
Games		*Solitaire Collection*
Discontinued	**Games**	*3D Pinball* *Chess Titans* *FreeCell* *Hearts* *InkBall* *Hold 'Em* *Purble Place* *Spider Solitaire* *Solitaire* *Tinker*
	Apps	ActiveMovie

		Anytime Upgrade Address Book Backup and Restore Cardfile CardSpace CD Player Chat Contacts Desktop Gadgets Diagnostics DriveSpace DVD Maker Easy Transfer Fax Food & Drink Help and Support Center Health & Fitness HyperTerminal Imaging Journal Media Center Meeting Space Messaging Messenger Mobile Device Center Movie Maker MSN Dial-up NetMeeting NTBackup Outlook Express Phone Companion Photo Gallery Photo Viewer Program Manager Steps Recorder Syskey Travel WinHelp Write
	Others	ScanDisk File Protection Media Control Interface Next-Generation Secure Computing Base POSIX subsystem

		HPFS Interix Video for Windows Virtual DOS machine Windows on Windows Windows SideShow Windows Services for UNIX Windows System Assessment Tool Windows To Go WinFS
Spun off to Microsoft Store		DVD Player File Manager Hover! Mahjong Minesweeper
Deprecated		Snipping Tool Media Player
Defunct		Internet Explorer Pay
Category List		

Windows command-line programs and shell builtins

- v
- t
- e

COMMAND.COM Command Prompt Windows PowerShell Recovery Console	
File system navigation	cd (chdir) dir popd pushd tree

File management	attrib cacls cipher compact copy del (erase) deltree icacls mkdir (md) mklink move openfiles recover ren (rename) replace rmdir (rd) robocopy takeown xcopy
Archiving	expand extrac32 extract makecab pax tar
Disk management	chkdsk convert defrag diskcomp diskcopy diskpart diskraid diskshadow drvspace fdisk format fsutil label manage-bde refsutil subst scandisk sys

	vol vssadmin
Processes	at exit kill powercfg runas sc schtasks shutdown start taskkill tasklist
Registry	assoc ftype reg regini regsvr32
User environment	chcp cmdkey date graftabl mode path set setver setx time title ver where whoami
File contents	comp edit edlin fc find findstr print type

Scripting	choice clip cscript doskey echo for forfiles goto if more pause prompt rem timeout
Networking	arp BITSAdmin cURL getmac hostname ipconfig nbtstat net netsh netstat nslookup PathPing ping rpcping route scp setspn sftp ssh ssh-add ssh-agent ssh-keygen ssh-keyscan tracert winrm winrs
Maintenance and care	auditpol dispdiag driverquery

	eventcreate eventtriggers logman mofcomp msiexec ntbackup pnpunattend pnputil REAgentC relog sfc sxstrace systeminfo tracerpt typeperf w32tm WBAdmin wecutil wevtutil winmgmt winsat wmic
Boot management	bcdedit bootcfg bootsect fixboot fixmbr
Software development	debug exe2bin QBasic
Miscellaneous	break cls dism dpath gpresult gpupdate help MSCDEX pentnt tpmtool tpmvscmgr wsl

List of DOS commands
Environment variables
Windows Support Tools

Common Language Infrastructure

- v
- t
- e

Architecture		Application domain Code Access Security Common Intermediate Language instructions Common Type System Platform Invocation Services Virtual Execution System
Components		Assembly Delegate Global Assembly Cache Manifest Metadata Standard Libraries
Implementations	**Microsoft**	.NET .NET Framework .NET Compact Framework .NET Micro Framework
	Other	Mono DotGNU
Languages	**Major languages**	C# Visual Basic F# PowerShell
	Other	Axum A# Boo Cobra C++/CLI

		IronScheme IronPython IronRuby JScript .NET J# Nemerle Oxygene Phalanger Q# Scala Small Basic X#
	Comparison	C# and Java C# and Visual Basic .NET Java and .NET platforms Visual Basic and Visual Basic .NET

Microsoft APIs and frameworks

- v
- t
- e

Graphics	Desktop Window Manager Direct2D Direct3D D3D (extensions) GDI / GDI+ WPF Silverlight WinUI Windows Color System Windows Image Acquisition Windows Imaging Component DirectX Graphics Infrastructure (DXGI) Windows Advanced Rasterization Platform WinG
Audio	DirectMusic DirectSound DirectX plugin XACT Speech API XAudio2

Multimedia	DirectX Media Objects Video Acceleration Xinput DirectInput DirectShow Image Mastering API Managed DirectX Media Foundation XNA Windows Media Video for Windows
Web	MSHTML RSS Platform JScript VBScript BHO XDR SideBar Gadgets TypeScript
Data access	Data Access Components (MDAC) ADO ADO.NET ODBC OLE DB Extensible Storage Engine Entity Framework Sync Framework Jet Engine MSXML OPC
Networking	Winsock LSP Winsock Kernel Filtering Platform NDIS Windows Rally BITS P2P API MSMQ MS MPI

	DirectPlay
Communication	Messaging API Telephony API WCF
Administration and management	Win32 console Windows Script Host WMI (extensions) PowerShell Task Scheduler Offline Files Shadow Copy Windows Installer Error Reporting Event Log Common Log File System
Component model	COM COM+ ActiveX Distributed Component Object Model .NET Framework
Libraries	Framework Class Library Microsoft Foundation Classes (MFC) Active Template Library (ATL) Windows Template Library (WTL)
Device drivers	WDM WDF KMDF UMDF WDDM NDIS UAA BDA VxD
Security	Crypto API CAPICOM Windows CardSpace

	Data Protection API Security Support Provider Interface (SSPI)
.NET	ASP.NET ADO.NET Remoting Silverlight TPL WCF WCS WPF WF
Software factories	EFx Factory Enterprise Library Composite UI CCF CSF
IPC	MSRPC Dynamic Data Exchange (DDE) Remoting WCF
Accessibility	Active Accessibility UI Automation
Text and multilingual support	DirectWrite Text Services Framework Text Object Model Input method editor Language Interface Pack Multilingual User Interface Uniscribe

Microsoft free and open-source software (FOSS)

- v
- t
- e

Overview	Microsoft and open source Shared Source Initiative

Software	**Applications**	Atom Conference XP Family.Show File Manager Open Live Writer Microsoft PowerToys Windows Calculator Windows Console Windows Package Manager Windows Terminal WorldWide Telescope XML Notepad
	Video games	Allegiance
	Programming languages	Bosque C# Dafny F# F* GW-BASIC IronPython IronRuby Lean P Power Fx PowerShell Project Verona Q# R Open Small Basic Online TypeScript Visual Basic
	Frameworks and development tools	.NET .NET Bio .NET Framework .NET Gadgeteer .NET MAUI .NET Micro Framework AirSim ASP.NET ASP.NET AJAX ASP.NET Core ASP.NET MVC

ASP.NET Razor
ASP.NET Web Forms
Babylon.js
BitFunnel
Blazor
C++/WinRT
CCF
ChakraCore
CLR Profiler
Dapr
DeepSpeed
DiskSpd
Dryad
Dynamic Language Runtime
eBPF on Windows
Electron
Entity Framework
Fluid Framework
Infer.NET
LightGBM
Managed Extensibility Framework
Microsoft Automatic Graph Layout
Microsoft C++ Standard Library
Microsoft Cognitive Toolkit
Microsoft Detours
Microsoft Enterprise Library
Microsoft SEAL
Mimalloc
ML.NET
mod_mono
Mono
MonoDevelop
MSBuild
MsQuic
Neural Network Intelligence
npm
NuGet
OneFuzz
Open Management Infrastructure
Open Neural Network Exchange
Open Service Mesh
Open XML SDK
Orleans
ProcDump
ProcMon
Python Tools for Visual Studio

	R Tools for Visual Studio RecursiveExtractor Roslyn Sandcastle SignalR StyleCop SVNBridge T2 Temporal Prover Text Template Transformation Toolkit TLA+ Toolbox U-Prove vcpkg Virtual File System for Git Visual Studio Code Voldemort VoTT Vowpal Wabbit Windows Communication Foundation Windows Driver Frameworks KMDF UMDF Windows Forms Windows Presentation Foundation Windows Template Library Windows UI Library WinJS WinObjC WiX XSP xUnit.net Z3 Theorem Prover
Operating systems	MS-DOS (v1.25 & v2.0) Barrelfish SONiC CBL-Mariner
Other	ChronoZoom Extensible Storage Engine FlexWiki FourQ Gollum Project Mu ReactiveX SILK

		TLAPS TPM 2.0 Reference Implementation WikiBhasha
Licenses	Microsoft Public License Microsoft Reciprocal License	
Forges	CodePlex GitHub	
Related	.NET Foundation F# Software Foundation Microsoft Open Specification Promise Open Letter to Hobbyists Open Source Security Foundation Outercurve Foundation	
Category		

Categories:

- Windows commands
- .NET programming languages
- Unix shells
- Windows command shells
- Dynamically typed programming languages
- Free and open-source software
- Interpreters (computing)
- Microsoft free software
- Microsoft programming languages
- Object-oriented programming languages
- Procedural programming languages
- Programming languages created in 2006
- Scripting languages
- Software using the MIT license
- Text-oriented programming languages
- Windows administration
- Formerly proprietary software

Hidden categories:

- Articles with short description
- Short description matches Wikidata
- All articles with unsourced statements
- Articles with unsourced statements from May 2013
- Articles with unsourced statements from April 2020
- Wikipedia articles needing clarification from January 2014

Navigation menu

Personal tools

- Not logged in
- Talk
- Contributions
- Create account
- Log in

Namespaces

- Article
- Talk

☐

Variants expanded collapsed

Views

- Read
- Edit
- View history

☐

More expanded collapsed

Search

Search Go

Navigation

- Main page
- Contents
- Current events
- Random article
- About Wikipedia
- Contact us
- Donate

Contribute

- Help
- Learn to edit
- Community portal
- Recent changes
- Upload file

Tools

- What links here
- Related changes
- Upload file
- Special pages
- Permanent link
- Page information
- Cite this page
- Wikidata item

Print/export

- Download as PDF
- Printable version

In other projects

- Wikimedia Commons
- Wikibooks
- Wikiversity

Languages

-
- Български
- Bosanski
- Català
- Čeština
- Dansk
- Deutsch
- Eesti
- Español
- Esperanto
-
- Français
-
- Bahasa Indonesia
- Italiano
-
- Jawa
- Nederlands
-
- Norsk bokmål
- Polski
- Português
- Русский
- Suomi
- Türkçe
- Українська
-
-

Edit links