


# Understanding important PowerShell concepts

08/23/20182 minutes to read 

## In this article

Output is object-based

The command family is extensible

PowerShell handles console input and display

PowerShell uses some C# syntax

The PowerShell design integrates concepts from many different environments. Several of the concepts will be familiar to people with experience in shells or programming environments. However, few people will know about all of them. Looking at some of these concepts provides a useful overview of the shell.

## Output is object-based

Unlike traditional command-line interfaces, PowerShell cmdlets are designed to deal with objects. An object is structured information that is more than just the string of characters appearing on the screen. Command output always carries extra information that you can use if you need it.

If you've used text-processing tools to process data in the past, you'll find that they behave differently when used in PowerShell. In most cases, you don't need text-processing tools to extract specific information. You directly access portions of the data using standard PowerShell object syntax.

# The command family is extensible

Interfaces such as **cmd.exe** don't provide a way for you to directly extend the built-in command set. You can create external command-line tools that run in **cmd.exe**. But these external tools don't have services, such as Help integration. **cmd.exe** doesn't automatically know that these external tools are valid commands.

The native commands in PowerShell are known as *cmdlets* (pronounced command-lets). You can create your own cmdlets modules and functions using compiled code or scripts. Modules can add cmdlets and providers to the shell. PowerShell also supports scripts that are analogous to UNIX shell scripts and **cmd.exe** batch files.

## PowerShell handles console input and display

When you type a command, PowerShell always processes the command-line input directly. PowerShell also formats the output that you see on the screen. This difference is significant because it reduces the work required of each cmdlet. It ensures that you can always do things the same way with any cmdlet. Cmdlet developers don't need to write code to parse the command-line arguments or format the output.

Traditional command-line tools have their own schemes for requesting and displaying Help. Some command-line tools use */?* to trigger the Help display; others use *-?*, */H*, or even *//*. Some will display Help in a GUI window, rather than in the console display. If you use the wrong parameter, the tool might ignore what you typed and begin executing a task automatically. Since PowerShell automatically parses and processes the command line, the *-?* parameter always means "show me Help for this command".

## Note

If you run a graphic application in PowerShell, the window for the application opens. PowerShell intervenes only when processing the command-line input you supply or the application output returned to the console window. It does not affect how the application works internally.

# PowerShell uses some C# syntax

PowerShell is built on the .NET Framework. It shares some syntax features and keywords with the C# programming language. Learning PowerShell can make it much easier to learn C#. If you're already familiar with C#, these similarities can make learning PowerShell easier.