What is PowerShell?

Definition of PowerShell

PowerShell is the shell framework developed by Microsoft for administration tasks such as configuration management and automation of repetitive jobs. The term 'PowerShell' refers to both – the shell used to execute commands and the scripting language that goes along with the framework.

The scripting aspect of it is similar to Perl programming. The shell is comparable to bash in UNIX, with Microsoft even incorporating commands such as man, ls and ps for convenience.

PowerShell 1.0 was released in November 2006 for Windows XP SP2, Windows Server 2003 SP1 and Windows Vista. While, initially, PowerShell had to be manually installed, the latest version 5.0 is available default with Windows 10. So, you can just go to Cortana and type 'PowerShell' or navigate from the Start menu. Read this to know more about which Windows version uses which PowerShell version.

PowerShell also comes with an Integrated Scripting Environment (ISE). The ISE screen is split into two parts – the top one is used to write the script and the bottom for running commands manually. The ISE gives you a GUI experience, with smart syntax suggestions, coloring, tab completion and error handling.

If you are a Windows administrator who has to perform user management, DNS configurations and other tedious tasks frequently, PowerShell is the tool for you.

Benefits of PowerShell over Command Prompt

Well, PowerShell certainly has more power! Command prompt is an interface available to execute simple DOS commands; most users have not explored it beyond ping, ipconfig or in the programming world, ftp. However PowerShell is much more than that. While there are many differences between the two, here are a few important ones:

1. **PowerShell uses cmdlets, not commands.** Now, cmdlets are not just a different way of calling the same thing, but they expose complex system administration functionalities such as registry management and Windows Management Instrumentation (WMI) to the user. This makes them far more effective than the command prompt.

2. **PowerShell is object-oriented.** The data output from cmdlets are objects (an example of how object-orientation makes PowerShell attractive) and not just text. This provides more flexibility to play around with complex data.

3. **PowerShell is developed using the .NET framework.** This allows PowerShell scripts to use .NET interfaces and extend features that are not provided by default through cmdlets. The other way around is also possible – embedding PowerShell scripts in .NET code.

More about Cmdlets

Cmdlets are lightweight commands used in the PowerShell environment. Most of the cmdlets in PowerShell use the Verb-Noun format.

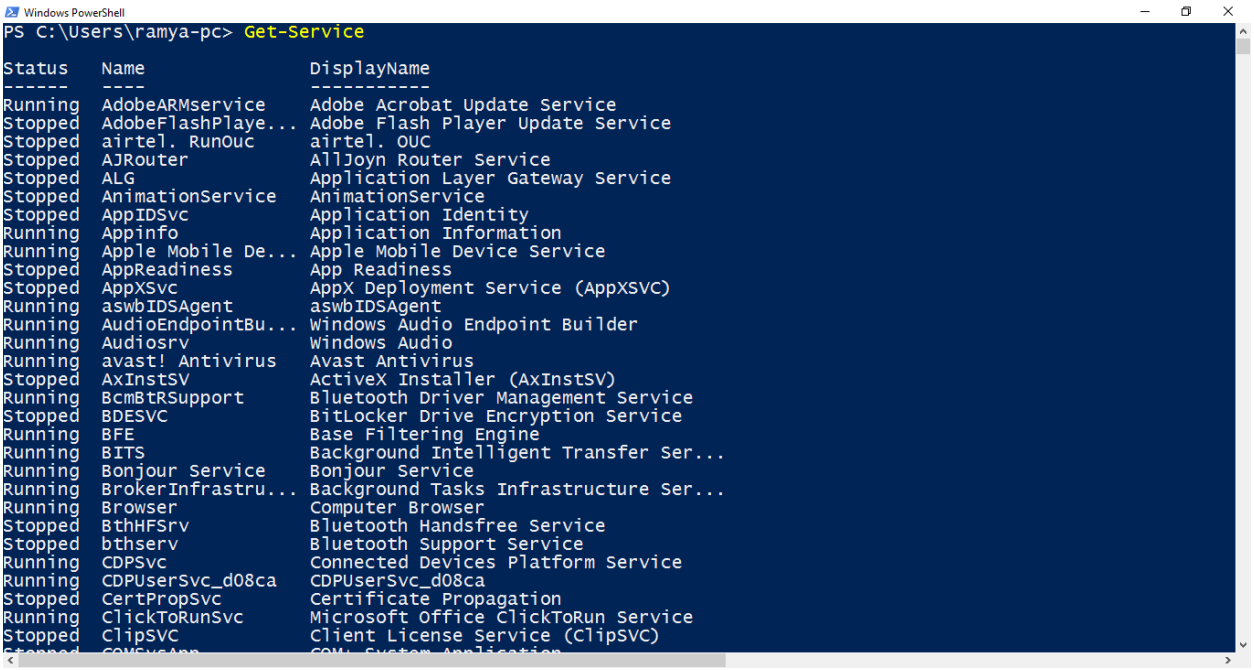
For example, Get-Command, Update-Help, Start-Service, etc.

To know more about cmdlets and understand what differentiates them from commands, you can read this link. One of the key differences that we will discuss below is its object orientation – it reads input from a pipe and outputs objects (not text) to a pipe.

Examples:

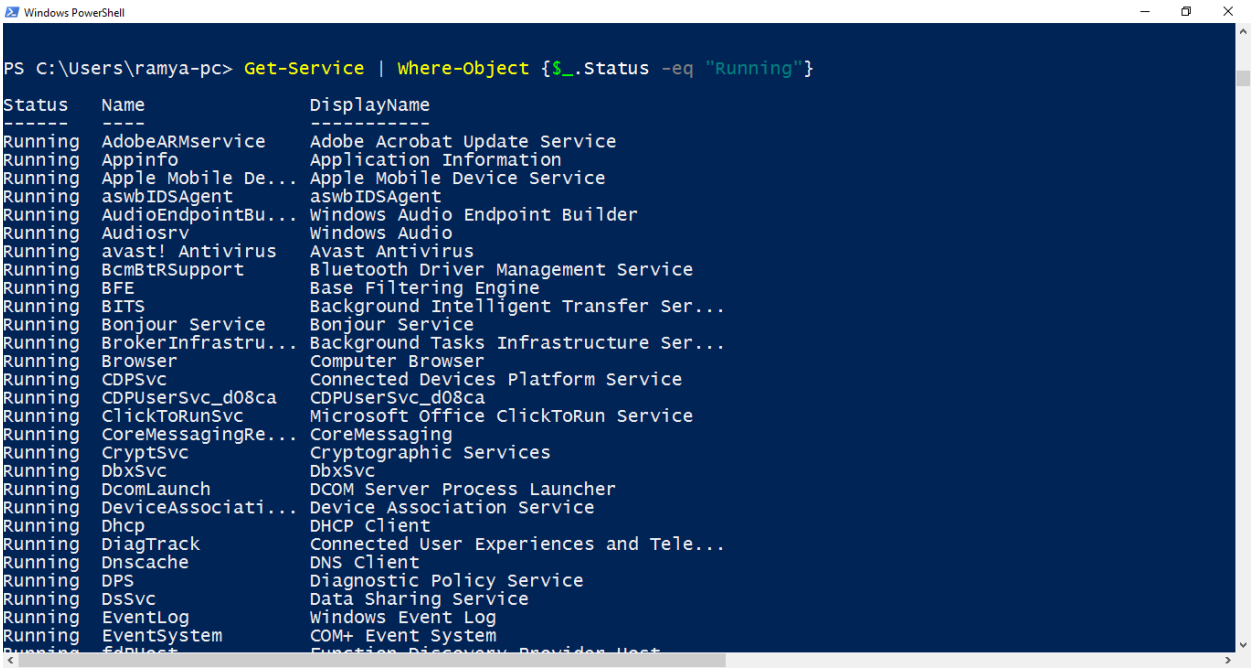
Let us go through an example to understand how this works.

On the command line, when you run the `Get-Service` cmdlet, you get a list of services on your machine.



You can further filter these to just show the services that are running:

```
Get-Service | Where-Object {$_.Status -eq "Running"}
```



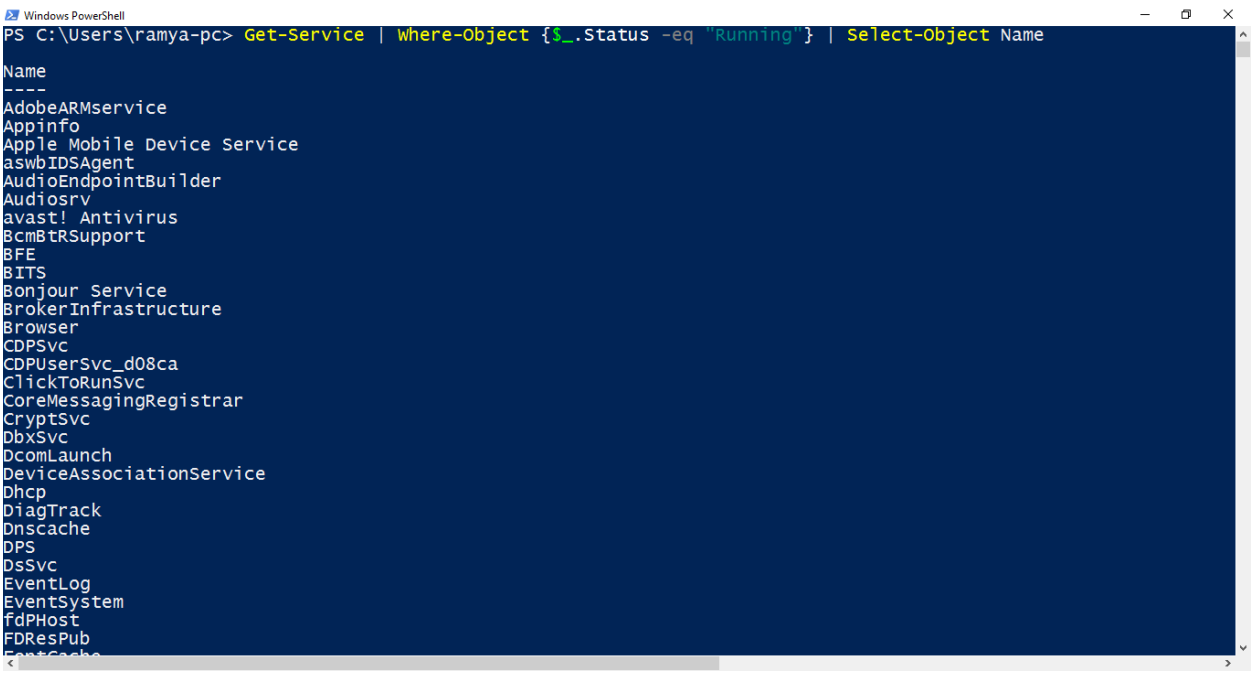
Here, PowerShell processes every record output that `GetService` throws, evaluates whether the ‘Status’ attribute is ‘Running’, and filters accordingly.

Note: $_ refers to the current record/object in the pipe.

One thing this example shows is the cmdlet’s record-oriented feature. The processing of one record at a time gives the flexibility to interpret data in a more intelligent way than it would be possible with text streams.

You can filter the previous output further, to just display the ‘Name’ of the running services.

```
Get-Service | Where-Object {$_.Status -eq "Running"} |
Select-Object Name
```



To learn more about any command, use

Get-Help –Name <Cmdlet name>

Or Get-Help –Name <Cmdlet name> -Online

The latter one opens up detailed help with examples on the msdn site.

Here, you can learn some basic commands to get you initiated into PowerShell. If you are a visual person, go for this video. For a detailed list, this would help.

Quick Intro to Aliases

Aliases are alternate names for your cmdlets. These are usually handy for frequently used commands and when you don't want to type the whole Verb-Noun format. For example, Get-Command has an alias 'gcm', Get-Service has 'gsv', Where-Object has a simple '?'.

Many UNIX commands are setup as aliases by default. For example, cat->Get-Content, man->help, ps->Get-Process, etc.

Run `Get-Alias` to see the complete list of aliases available in PowerShell.

Of course, you can create your own aliases as well.

Creating a PowerShell Script

To create a PowerShell script, all you have to do it open a file, write your code and then save it. PowerShell scripts have a .ps1 extension. Your script can then be run manually or

automated to run as a job every day to perform administration tasks. Get started here for creating simple scripts with looping and conditions.

Just the Tip Of The Iceberg

Through PowerShell commands and scripts, there is so much benefit to be gained for an IT administrator. Here is a list of use cases where an administrator can leverage PowerShell commands. From gathering information about servers to managing folders, processes, services, memory, network, software installations and registries, there are tons of features that PowerShell encapsulates. Not to forget about its seamless integration with .NET. Once you get over the initial learning curve, you can really start exploiting the functionalities of PowerShell.