PowerShell Core 6.2

Setup, Syntax, CLI, CmdLets, Control Flow, DSC, Modules, Security, Remoting, .NET Integration

PowerShell Core 6.2 is Microsoft's latest innovative scripting and automation engine. The older versions of PowerShell (5-1) were based on the .NET Framework and only ran on Windows. In contrast, the new PowerShell Core 6.x which this course covers is based on .NET Core and so can run on Windows, Linux and macOS. Both PowerShell Core and .NET Core are open source. PowerShell Core is the basis for Azure Cloud Shell. PowerShell Core commands have a very flexible syntax and can be executed immediately or stored in a script, and later executed on the local machine or (after appropriate security steps) on remote machines.

PowerShell Core commands are best created via the PowerShell extension to Visual Studio Code; they can also be created via any text editor.

PowerShell has a few key concepts that separate it from previous shell languages. It is based on .NET and its syntax borrows from C#, so moving between both is quite easy. Like all shells, PowerShell works on the basis of a pipeline. Unlike other shells, what flows along the PowerShell pipeline are .NET objects. PowerShell offers the idea of consistently named cmdlets (<verb>-<noun>: e.g. `Get-Process` lists processes).

Target Audience

System administrators and software developers wishing to benefit from the modern approach to scripting on Windows, Linux and macOS.

Prerequisites

Good knowledge of scripting with any similar environment (e.g. Bash).

Contents of One-Day Training Course

Tour of PowerShell Core Ecosystem

- Overview of all parts of PowerShell Core
- Installing and configuring
- Choosing an editor
- Getting started with your first script
- Exploring main language features

CmdLets

- CmdLets (Command lets, as in, small commands) are individual pieces of functionality that can be used standalone or more usually combined to form larger scripts – we explore how they work
- Best practices for creating CmdLets

Variables

- Think of PowerShell as a simple program -ming language, so variables needed
- Scope. naming, usage

More Language Constructs

- Control flow
- Functions
- Exception handling with try / catch
- Error info within exceptions using `$_`

Shell Constructs

- Formatting output from within script
- Accepting command line arguments
- Environment
- OS interaction

Providers

- To extend the range of data stores exposed to PowerShell scripts, you can create and load custom providers
- Architecture of a provider+sample impl

Server Automation

- PowerShell Core works with many kinds of server automation packages
- DSC Core and DSC Resource Kit
- Chef, Ansible, Puppet
- Scripting in Azure Cloud Shell

Security

- Use of SSH as protocol for PSRP
- Signing, execution policy, authenticode

PowerShell Core And OpenAPI

- Review of use of OpenAPI as a very popular definition syntax for precisely describing REST API
- Using AutoRest to generate CmdLet client

Remoting

- WSMan
- Windows management Framework
- PowerShell Remoting Protocol: PSRP

.NET Extensions

- Using PowerShell Core to call a standard .NET Core assembly and also calling into third party .NET assemblies
- Designing and building our own custom .NET assemblies that can be used from PowerShell Core
- Architectural guidance for how to structure larger integration

Project

- Many larger commercial server products are now being delivered with a PowerShell interface (e.g. Vmware's PowerCLI). We explore how to design something similar for our own sample server product