PowerShell Advanced Cookbook

Enhance your scripting skills and master PowerShell with 90+ advanced recipes

Morten Elmstrøm Hansen

www.bpbonline.com

First Edition 2024

Copyright © BPB Publications, India

ISBN: 978-93-55516-732

Dedicated to

*My beloved wife **Janni**, my sons **Theo-Maximillian** and*
***Milo-Matthæus**, my family, my in-laws,*
my friends and my colleagues at Energi Danmark

About the Author

Morten Elmstrøm Hansen is a dedicated IT professional and entrepreneur. For the past eight years he has been working at Energi Danmark, one of Denmark's leading energy companies. He plays a central role in the IT operations department where he leverages his expertise in PowerShell development, DevOps and server administration to monitor, optimize and automate essential processes, ensuring operational continuity. Morten is proficient in containerization and kubernetes as well as cloud technologies and he utilizes his skills to be the forefront for larger projects involving such cutting-edge technologies.

His professional qualifications include a certification in PowerShell Administration, an AP degree in IT technology and additional courses in digital marketing. Morten also holds a mix of several other technical and commercial educations such as a business college education that provides a solid foundation in commerce and administration, with a focus on economics, sales, communication and IT. Furthermore, he has a higher technical education that combines technical and scientific subjects with general academics. Additionally, he pursued a higher preparatory examination focusing on mathematics and psychology, further preparing him for his academic and professional pursuits in technology.

In addition to his role in IT, Morten co-owns and serves as the part time CFO of a family cleaning business. He also owns and manages an online store, where he applies his skills in digital marketing, IT and economics. His diverse roles demonstrate his ability to navigate and excel in multiple fields, from IT management and development to business operations.

On a personal note, Morten is a devoted family man, living with his wife and two youngest children, on the outskirts of one of Denmark's largest cities.

About the Reviewers

- **Lasse Hein Olesen** is an IT Service Engineer, working within the MES and automation field, with an expertise in troubleshooting and problem solving. He is passionate about anything IT related and is always interested in learning something new and finding new challenges. He actually learned PowerShell from Morten, the author of this book. It is his first time being a technical reviewer. He's currently focusing on advancing his PowerShell skills and expanding/building up his homelab. He is hoping to explore DevOps and Kubernetes in the near future.

- **Tibor Soós** is a freelance PowerShell developer and trainer, having over 25 years of extensive experience in automating Windows, Active Directory, Exchange, Azure AD, Jira Service Desk and various Identity and Access Management systems like Saviynt, CyberArk, SecZetta. He has contributed his expertise to industry-leading enterprises, showcasing a deep understanding of PowerShell as a former MVP and the author of a Hungarian-language PowerShell book.

  Tibor is committed to empowering IT professionals with his wealth of knowledge and his enterprise-grade productivity modules, offering guidance and best practices to enhance their scripting and automation capabilities.

Acknowledgement

My deepest gratitude and heartfelt appreciation to my wife Janni Elmstrøm Hansen and our two boys Theo-Maximillian and Milo-Matthæus. Without you and your continuous love and support, this book would never have become a reality. Even during moments of doubt and harsh times, you are always there to pick me up and make sure I follow my heart and my dreams. You will always be the love of my life and the soul of my being.

I am also deeply grateful for the never-ending love and support from my family: My dad Johnny, my mom Lene, my brother Mads, and my sister Julie and for my wife´s family. Without you I would not be the person I am today. You have always picked me up, held me when needed and supported me, especially through the tough times in life.

I would also like to thank and acknowledge all my close friends in life and all my colleagues at Energi Danmark. Thank you for your invaluable support and for always believing in me. You have all been instrumental in my personal and professional development throughout the years, and you have taught and given me more than I could have ever asked for.

Additionally I would like to extend my sincerest appreciation to the reviewers of this book, Lasse Hein Olesen and Tibor Soós. Your input and attention to detail have been invaluable in creating this book and i am deeply grateful for your dedication and support.

Finally, I would like to thank BPB Publications for this incredible opportunity and for all the help and support throughout the journey of making this book become a reality.

Preface

PowerShell is a powerful scripting language, automation framework and command-line shell developed by Microsoft that is built on the .NET framework. It is an essential tool because it allows system administrators and developers to automate and optimize complex administrative tasks across multiple systems efficiently. PowerShell's deep integration with Windows and other Microsoft products makes it an invaluable tool for administrating, managing and optimizing Windows environments.

This book is intended for developers and system administrators with a novice or intermediate understanding of PowerShell who are looking to advance their skills. It is also beneficial for experienced professionals seeking to enhance their existing PowerShell capabilities.

Designed as a cookbook, this book enables readers to expand and build upon their current PowerShell knowledge and skillset. Topics covered in detail include creating PowerShell unit tests using Pester, managing and administrating Azure and AWS cloud services, remote script execution, Active Directory management, PowerShell desired state configuration and more. Each chapter includes recipes that delves into the topics, accompanied by code examples and walkthroughs. After reading this book, readers should have gained knowledge and skills that enables them to build better and more advanced scripts and applications while also understanding key principles of automation and optimization. This will also enable the readers to streamline processes and enhance administrative tasks more efficiently using PowerShell.

Chapter 1: Introduction to Advanced PowerShell Concepts – Explores advanced PowerShell concepts, scripting best practices, and the configuration and setup of the Visual Studio Code **Integrated Development Environment (IDE)**. The chapter also makes a short introduction to the different PowerShell versions available and furthermore, it introduces PowerShell providers which are used to facilitate and provide access to data across different data stores.

Chapter 2: Advanced PowerShell Functions – Delves into advanced PowerShell functions and covers a detailed explanation of its structure and capabilities, including dynamic parameters, parameter sets and lifecycle events. It also covers key elements such as the CmdletBinding attribute, parameter validation and output formatting. Furthermore, it explains how to implement handling for pipeline input and the ShouldProcess capability. The chapter also introduces object-oriented PowerShell concepts in the form of PowerShell classes. It offers a step-by-step guide to constructing classes with properties and methods.

Chapter 3: Flow Control and Looping – Is about using different methods for controlling and optimizing the flow of script execution. It introduces the reader to conditional statements using comparison and logical operators. It deeply covers advanced looping techniques such as nested loops, labelled breaks and continue statements, the use of retry logic in loops and the use of pipeline processing and also the Foreach-Object loop for iterating collections. Additionally, it explains how to utilize the switch statement to handle multiple conditions and it delves into using script blocks for dynamic and flexible flow control in terms of callbacks and event handlers. Furthermore, the chapter introduces the reader to multi-threading in the forms of parallel processing using PowerShell jobs and non-sequential processing using the Foreach-Objects parallel parameter.

Chapter 4: Error Handling – Dives deep into error handling in PowerShell and introduces the reader to topics such as handling different types of errors, implementing error handling blocks in advanced scenarios and how to use the ErrorAction preference variable and the ErrorAction parameter. It also showcase how to create custom error classes for enhanced error reporting and how to handle errors in background jobs.

Chapter 5: Scripting Techniques – Focuses on a range of different scripting techniques that are common among different programming languages but with a focus on such techniques tailored specifically for PowerShell. The chapter covers topics such as script parameters, parameter validation, string manipulation and formatting techniques and scripting for cross-platform compatibility. Additionally, it introduces the reader to PowerShell execution policies and script signing for enhancing script security. Furthermore, the chapter dives deeper into the creation of PowerShell modules and repositories which are essential for organizing and distributing PowerShell code efficiently.

Chapter 6: Remote Script Execution: PowerShell Remote Management – Guides the reader through setting up, configuring, and securely managing remote PowerShell sessions using PowerShell remoting and Windows remote management. It explains how to use session configurations to restrict and grant privileges and permissions for specific users and groups on remote hosts. The chapter also provides a more in-depth insight into securing and authenticating remote sessions using credentials, encrypted XML files, the Windows credential manager and also how to configure secure and encrypted certificate-based authentication.

Chapter 7: Testing with Pester – Introduces the reader to the Pester PowerShell testing framework. The chapter will guide the reader through setting up the Pester framework and it dives into Pesters structure and components. The chapter covers how to create unit tests for PowerShell functions, how to group and organize tests and how to mock dependencies. Furthermore, the chapter will provide strategies for testing infrastructure components and also how to implement code coverage analysis in tests.

Chapter 8: Working with XML and JSON – Presents a detailed introduction to XML and JSON that are used for representing and structuring data in formats that are both human readable and easy for scripts and applications to interpret. While the main focus is targeted at XML, the reader will learn how to read and write XML files using cmdlets, the XML accelerator and by using .NET classes, but the reader will also learn how to read and write JSON files and how to convert between JSON and PowerShell objects. It also dives into querying and extracting data from XML files using XPath expressions and how to serialize and deserialize PowerShell objects using CliXml.

Chapter 9: Active Directory Management - Delves into the essentials of managing and automating Active Directory tasks using PowerShell. This chapter introduces the ActiveDirectory module which is an essential tool in PowerShell for interacting with AD environments. The reader will learn how to manage AD users and groups efficiently. The chapter also introduces techniques for performing bulk operations, which are crucial in larger organizations where managing numerous accounts manually is impractical. Furthermore, it explores how to query and filter AD objects effectively, using PowerShell's filtering capabilities to streamline administrative Active Directory tasks.

Chapter 10: Managing Azure with PowerShell – Examens Azure cloud services management using the Azure command line interface (AzureCLI) and PowerShell. This chapter covers working with and managing Azure virtual machines, storage accounts, blobs, and file shares. It also dives deeper into Azure EntraID and focuses on the creation and management of users, groups, and resource access permissions. Additionally, it provides insight into automating resource provisioning and management focusing on creating service principals that are used for programmatically connecting to Azure using scripts and also how to create scripts that are used for configuration and automatic provisioning of Azure resources.

Chapter 11: Managing AWS with PowerShell – Walks through AWS cloud services management and introduces the AWS tools for PowerShell. It describes how to install and configure these tools and also how to configure credentials for accessing AWS programmatically. The chapter dives deeper into AWS **identity** and **access management (IAM)** showcasing how to manage IAM users and groups and how to create and manage access keys, permissions, and policies. Furthermore, the chapter not only describes how to create and manage EC2 instances, key pairs and security groups but also how to manage S3 buckets and how to upload and download objects to such buckets.

Chapter 12: Microsoft 365 Applications Management – Showcases how to install and use specific application modules that focus on different aspects of Microsoft 365 applications management. The chapter covers management of SharePoint online, Exchange online and Microsoft Teams using PowerShell. Additionally, it introduces the Microsoft Graph API and the Microsoft Graph PowerShell SDK module describing installation, configuration, and authentication. Furthermore, the chapter gives an overview of how to manage Entra ID users and licenses using PowerShell and the modules.

Chapter 13: Desired State Configuration – Provides an extensive insight into desired state configuration with PowerShell and outlines how to write and apply meta configurations from a centralized management server to remote nodes and DSC configurations on remote target nodes. This includes configuring local configuration managers, creating DSC configurations for managing infrastructure and using public resource modules. The chapter also describes how to remove resources and configurations and how to handle failed configurations.

Chapter 14: Managing Windows Components – Is about using PowerShell to manage different Windows server and workstation components. The chapter covers how to create, manage, and delete windows services, how to start and manage processes, how to manage and configure network settings and how to initialize, partition and format disks. It also dives deeper into firewall rules and the Windows task scheduler.

Chapter 15: SAPIEN PowerShell Studio IDE – Explores the advanced PowerShell IDE created by SAPIEN Technologies. This IDE takes PowerShell scripting to a new level and enables you to not only create advanced scripts from templates but also to easily create and compile sophisticated GUI applications, Windows services, and packaged executables from your PowerShell scripts. This chapter introduces the IDE and demonstrates how to use it to create GUI applications and Windows Services. Furthermore, it depicts how to compile scripts into executables and how to create MSI installers for executables using the packager and installer managers. It also peeks into more applications created by SAPIEN Technologies such as the PowerShell module manager and the VersionRecall versioning and backup tool.

Code Bundle and Coloured Images

Please follow the link to download the
Code Bundle and the ***Coloured Images*** of the book:

https://rebrand.ly/q7m2itb

The code bundle for the book is also hosted on GitHub at **https://github.com/bpbpublications/PowerShell-Advanced-Cookbook**.
In case there's an update to the code, it will be updated on the existing GitHub repository.

We have code bundles from our rich catalogue of books and videos available at **https://github.com/bpbpublications**. Check them out!

Errata

We take immense pride in our work at BPB Publications and follow best practices to ensure the accuracy of our content to provide with an indulging reading experience to our subscribers. Our readers are our mirrors, and we use their inputs to reflect and improve upon human errors, if any, that may have occurred during the publishing processes involved. To let us maintain the quality and help us reach out to any readers who might be having difficulties due to any unforeseen errors, please write to us at :

errata@bpbonline.com

Your support, suggestions and feedbacks are highly appreciated by the BPB Publications' Family.

Table of Contents

Chapter 1
Introduction to Advanced PowerShell Concepts

Introduction

This chapter of the book introduces you to the concepts of advanced PowerShell and advanced scripting best practices. It will also cover the basics of setting up the **integrated development environment** (**IDE**) used in this book. We are primarily using the Visual Studio Code IDE with PowerShell extension, but another more advanced PowerShell specific IDE will also be introduced and will be covered more in detail in another chapter. In this chapter we will also look at the different newer PowerShell versions like PowerShell 5, PowerShell 7, and PowerShell Core. Lastly this chapter will make an introduction to the different PowerShell providers that can be used to navigate and leverage the different data stores such as the filesystem, registry, certificate, environment and so on. Most of the code and examples in this book can be used in the different mentioned versions of PowerShell, but some examples might need a specific version like PowerShell 5 or PowerShell 7. In our examples we are (primarily) using PowerShell 7 unless else is specified. If a specific PowerShell version is needed for a specific code example or recipe, it will be mentioned in detail regarding that specific context. Unless else is mentioned specifically, the recipes and examples in this book are created in Windows 10 (And should also be applicable in Windows 8 and Windows 11).

Structure

The chapter covers the following topics:

- Introduction to advanced PowerShell concepts
- PowerShell scripting best practices
- PowerShell integrated development environment
    - **Recipe 1:** Install and set up Visual Studio Code with PowerShell Extension
- Advanced Sapien PowerShell Studio IDE
- PowerShell versions
- PowerShell providers
    - **Recipe 2:** Use providers to access data stores

Objectives

After this chapter you will know more about advanced concepts of PowerShell and know more about what is needed to leverage its full potential to automate complex tasks, managing different systems, streamline administrative processes and build better and more advanced scripts. You will have learnt about common best scripting practices that makes sure your scripts are reliable and efficient and that you keep a certain standard when creating scripts. You will be introduced to different IDE´s and how to setup the VS Code IDE, which is used in this book, and how to utilize this to write scripts and run PowerShell commands directly in the IDE. You will also have been introduced to the more advanced PowerShell IDE, Sapien PowerShell studio, about which we will cover in more detail in a later chapter. Furthermore, you will have been introduced to different PowerShell versions, and learnt how they differ and what the use cases are for these versions. Lastly, you will also have learnt about the PowerShell providers and how to utilize these to access and work with the data in different data stores such as the Windows Registry, the filesystem, certificate stores and more.

Introduction to advanced PowerShell concepts

Advanced PowerShell involves the use of powerful features and techniques that extends beyond the main usage of the PowerShell scripting language. These concepts are for more experienced and expert users who would want to use the full potential of PowerShell for tasks like automation, managing systems and greatly improving scripts. Some of the key aspects of advanced PowerShell are to build modular scripts instead of building simple scripts. The case of breaking down a script into modular and reusable components makes it much more maintainable. This can be broken down to practically defining advanced functions, creating modules, and organizing your scripts into several files to make them more manageable. This will also make it a lot easier to promote code reuse and help in collaboration between different developers.

An example would be if you have created a function in a script that could be reused in other scripts, then instead of having to copy-paste the function into these, you would either create a file containing that function, or even better, to make it more advanced you would create a module and add the function to that module instead for re-usability. Then you would only have to install that specific module on your systems and import it, so you would be able to use that function in every scope and context whenever you need it just by calling it, since it would then already be imported into that scope or context. This is the same principle as with built-in modules and **`cmdlet`**s that are available to you when you open a new PowerShell session, here it is only your own custom function that are available and can be reused whenever needed, without having to copy-paste or importing it from a specific file every time you would have to use that function. Additionally, you would only have to update and make changes to the function in one place, the module, instead of having to handle multiple copies of a file, in different script paths, where the function file might have been copied too, and then must update the function in each of these files. Just by utilizing this method you would already have begun to make optimizations to your scripts and making use of advanced concepts.

Another necessary aspect of advanced PowerShell is the use of error handling and managing exceptions. Basic PowerShell scripts often lack extensive error handling which is crucial to building a robust script. Implementing advanced error handling mechanisms into scripts greatly improves the reliability and resilience of the script. This includes the use of try-catch blocks to catch and handle exceptions, logging errors for troubleshooting purposes, and displaying user-friendly error messages. Pipeline processing, **object-oriented programming (OOP**), PowerShell remoting, Classes, workflows, jobs, **desired state configuration (DSC)** and integrating and interacting with other technologies and systems are other fundamental concepts when it comes to advanced PowerShell. This book will cover the various advanced features and concepts that PowerShell offers and when combining the different knowledge from these advanced concepts, you will be able to create more advanced scripts and even write programs using PowerShell, that can be used to greatly optimize and automate not only your scripting and development tasks, but also improve and streamline a lot of your current scripts that you may already have to use in your daily work.

PowerShell scripting best practices

To make sure your scripts and your programs are reliable and efficient it is always a good idea to follow some best practices. Even though best practices are exactly that, *best practices*, meaning they are not resolute and different people have different opinions on what these are, there are still a lot of commonalities between them. These are the following practices I consider to be essential to write powerful and optimal scripts and programs:

- **Use readable and self-documenting code**: Your code should be easy to read and should be understandable. You should use meaningful names for variables, functions, classes, and other elements. You should provide clear comments to explain important details and more complex logic. It is also a good idea to document your scripts with details about its purpose and more specific usage instructions, especially if they should be shared with others.
- **Break your scripts down into reusable functions and modules**: This improves the organization of the code, encourages code reuse and in general simplifies the overall maintenance of your scripts. Each function and class must have a clear purpose and must be designed to perform specific tasks.
- **Gracefully handle errors**: This means to implement proper error handling in your scripts. You should use try-catch-finally blocks to catch and handle exceptions, log errors for troubleshooting purposes and return descriptive error messages to the users. Implementing effective error handling will significantly improve the reliability and resilience of your scripts.
- **Validate input parameters to ensure the stability and security of your scripts**: You should use parameter validation to make sure that constraints are set on input values, especially from users. This can prevent unexpected behavior and script failures and ensures that input values are provided as the correct data types.
- **Use pipeline processing to streamline the processing of data:** Make sure to use `cmdlet´`s and functions that supports pipeline input and output to use filtering and sorting of the data. This also reduces the use of complex loops since this will be handled by the pipeline processor instead.
- **Test and debug your scripts, in different scenarios and environments**: Make use of the built-in functionalities for testing and debugging, like using breakpoints and making unit tests for your functions and classes. This makes it a lot easier to catch issues early in the process and makes your scripts more reliable. You should set up different test scenarios and make extensive testing in different environments.
- **Use a main function**: It is not required in PowerShell like it is in some other programming languages, it would still be a good idea to use a main function in your code. This gives a clear starting point and better control of the flow in your scripts. It also helps you to avoid polluting the global scope and makes it generally more readable. A lot of PowerShell developers do not use a main function in their scripts, but personally I prefer this method, especially when creating Windows services with PowerShell, where the services main loop is an encapsulated function inside the script itself, making it impossible for the function to call the running service script itself from inside that function, at least without a lot of issues and errors, using a main function would solve this. For simple and smaller scripts, adopting the use of a main function might not make much sense and can be omitted. It is generally up to the user to decide whether to use a main function,

but as with the Windows PowerShell service, in some cases it is unavoidable and should then be implemented as a default best practice to follow.

- **Performance optimize your scripts**: Especially when dealing with larger sets of data. Here techniques like parallel processing, caching, and optimizing your loops can greatly improve the performance and minimize execution time. If you are using PowerShell 7, implement the use of the *-parallel* switch when using `For Each-Object`. Go through your code and figure out if and where you can optimize the code, can you replace loops with piping something instead and so on.
- **Versioning control**: Another valuable practice is to implement the use of a versioning control system, like GIT. This makes it a lot easier to work together with other developers but also gives you a history of the changes in your scripts and makes it easier to roll back to a previous version if needed. It is also a great method for having backups of your scripts and their different versions.

By following these or at least some of these best practices, you should be able to create scripts that are much more reliable and efficient. And as you are getting better and more experienced in your coding you will eventually adopt your own best practices for your own specific needs. Through this book, we will investigate several topics and recipes that will incorporate these practices for creating efficient and advanced scripts and using techniques that are based on these practices used in different scenarios.

PowerShell integrated development environment

An **integrated development environment** (**IDE**) is an invaluable tool when it comes to writing advanced and powerful PowerShell scripts, and in software development in general. Without the benefits of the advanced capabilities these tools offer, it could be quite cumbersome to write efficient scripts and programs. An IDE provides a centralized workspace for coding that combines a code editor with debugging capabilities and other essential functionalities into a single program. Some features of an IDE include the ability to highlight syntax, code completion, formatting, line numbering and most of them also provides an integrated terminal or console so you can run commands directly from the IDE without having to use and switch to external tools. These are just the most basic features that an IDE includes, each has its own strengths and weaknesses, it all depends on the preferences of the individual user. Some IDE´s can be used for multiple languages like the Visual Studio Code editor and other IDE´s are meant for more specific languages like the Sapien PowerShell studio which is built solely for PowerShell.

The Visual Studio Code editor is a more lightweight and versatile editor that is free to use both privately and commercially. This editor can also be used on both Windows, Linux, and Mac, but its strength comes in all the extensions that are publicly available, which supports almost every language that you can think of. This makes it a valid choice for