PowerShell best practices

Arnaud PETITJEAN
apetitjean@start-scripting.io

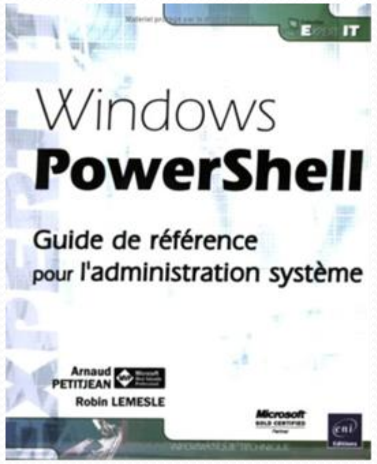
Arnaud Petitjean

@apetitjean

apetitjean@start-scripting.io

- PowerShell consultant and trainer since 2007
- 25 years' experience in IT
- Author of numerous books
- International speaker
- MVP since 2008
- Founder of the PowerShell Francophone community (https://powershell-scripting.com)
- Founder of Start-Scripting (https://start-scripting.io)

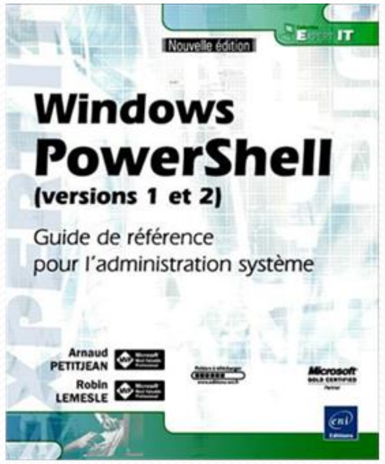
Arnaud Petitjean - Books


Windows PowerShell

Guide de référence pour l'administration système

Arnaud PETITJEAN

Robin LEMESLE

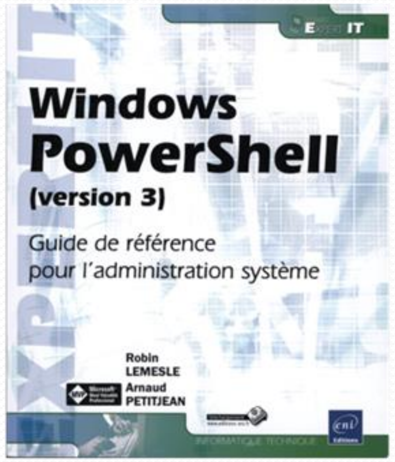

Nouvelle édition

Windows PowerShell (versions 1 et 2)

Guide de référence pour l'administration système

Arnaud PETITJEAN

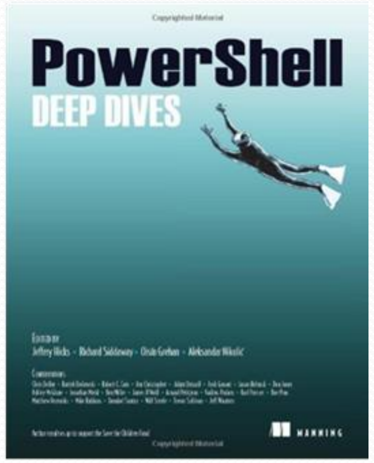
Robin LEMESLE


Windows PowerShell (version 3)

Guide de référence pour l'administration système

Robin LEMESLE

Arnaud PETITJEAN


PowerShell DEEP DIVES

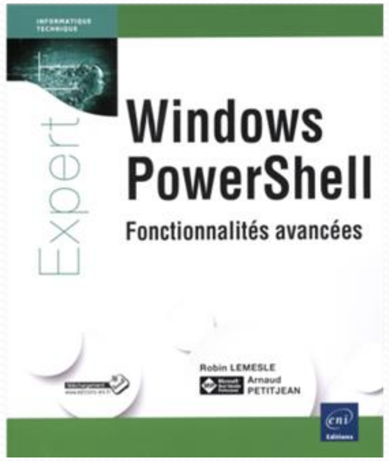
MANNING

Windows PowerShell

Les fondamentaux du langage

Arnaud PETITJEAN

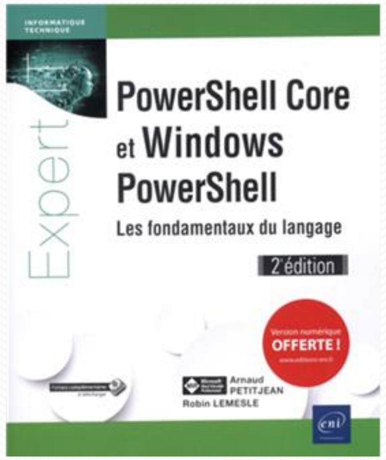
Robin LEMESLE


Windows PowerShell

Fonctionnalités avancées

Robin LEMESLE

Arnaud PETITJEAN


PowerShell Core et Windows PowerShell

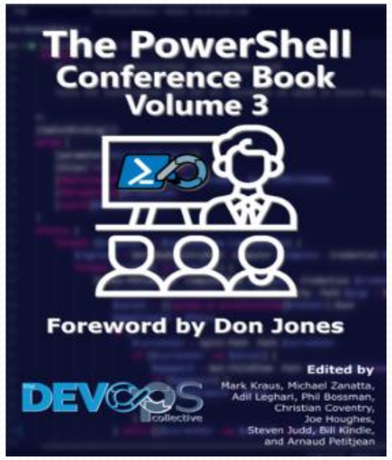
Les fondamentaux du langage

2^e^ édition

OFFERTE !

Arnaud PETITJEAN

Robin LEMESLE


The PowerShell Conference Book Volume 3

Foreword by Don Jones

Edited by Mark Kraus, Michael Zanatta, Adil Leghari, Phil Bossman, Christian Coventry, Joe Houghes, Steven Judd, Bill Kindle, and Arnaud Petitjean

DEVOPS collective

Best practices

- Initialize variables before using them
- Use **`Set-StrictMode`**
- Give a type to each of your parameters
- Do not accept passwords in clear text, but support the **`PSCredential`** type
- Do not store passwords in a script
- Do not use the **`Invoke-Expression`** command
- Indent your code
- Comment code

Best practices

- Give variables clear names (in English)
- A variable must never change his type during its lifetime
- Make use of advanced functions
- Never code function input data validation yourself
- Make parameters mandatory and set default values
- Add help to your functions (structured comments at least)
- Create modules and version them
- Only use approved verbs

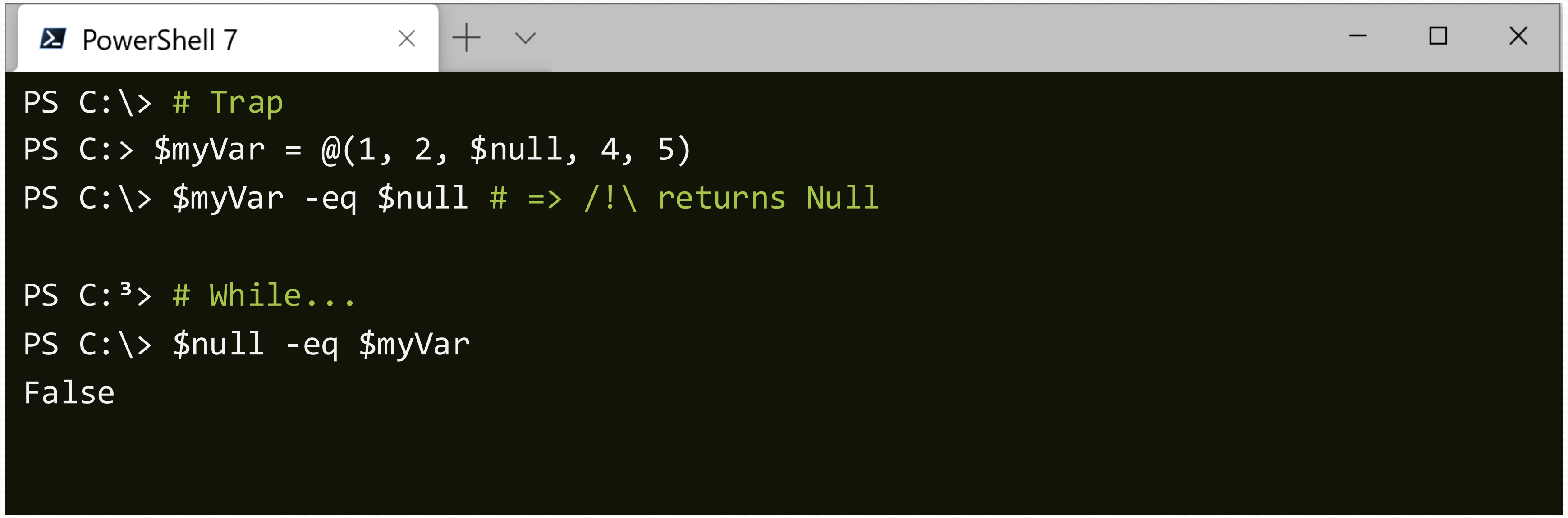
Best practices

- Use `Write-Verbose`, `Write-Debug`, and `Write-Error`
- Use single quotes if double quotes are not needed
- Give understandable names to parameters (without using plurals)
- Use the `#Requires` directive

Best practices

- Managing errors
- Test, test and test the code! TESTING can be a valuable ally
- Developing in TDD mode
- Do not use `Write-Host`
- Never use aliases in scripts
- Naming parameters when using them
- Compare $null -eq $myVariable rather than the other way round

Test the nullity of a variable



```
PS C:\> # Trap
PS C:> $myVar = @(1, 2, $null, 4, 5)
PS C:\> $myVar -eq $null # => /!\ returns Null

PS C:³> # While...
PS C:\> $null -eq $myVar
False
```

PowerShell Script Analyzer

PSScriptAnalyzer module

- Module designed to help improve code quality
- Can analyze DSC scripts, modules and resources
- Based on a set of predefined rules
- Rules based on best practices identified by Microsoft and the PowerShell community
- Generates a result object (errors and warnings) and points out potential improvements and problems
- Types of feedback: uninitialized variables, plaintext passwords, Invoke-Expression, etc.

PSScriptAnalyzer module

- Installation:

```
Install-Module -Name PSScriptAnalyzer
```

- How to use :
  - **`Get-ScriptAnalyzerRule`**: List rules
  - **`Invoke-ScriptAnalyzer`**: Launches analysis

https://github.com/PowerShell/PSScriptAnalyzer