Contributors:
Josh King
Josh Duffney
Adam Bertram
Mike Kanakos
Jonathan Medd
Thomas Lee
Prateek Singh
Dave Carroll
Dan Franciscus
Jeff Hicks
Tommy Maynard

A Blogger's Guide to Getting Started with PowerShell 7

#PS7Now

A Blogger's Guide to Getting Started with PowerShell 7

Jeff Hicks

This book is for sale at http://leanpub.com/ps7now

This version was published on 2020-03-26

This is a Leanpub book. Leanpub empowers authors and publishers with the Lean Publishing process. Lean Publishing is the act of publishing an in-progress ebook using lightweight tools and many iterations to get reader feedback, pivot until you have the right book and build traction once you do.

Contents

About This Book

This book is compilation of material published online centered on the general release of PowerShell 7. For this project, a number of PowerShell-focused bloggers and authors created content on many of the new features of PowerShell 7. The intent was to raise awareness of the new release and provide an incentive for existing Windows PowerShell administrators to make the move to PowerShell 7. The content was released over the course of week in March 2020 using the `#PSBlogWeek` and `#PS7Now` tags on Twitter. If you missed some or all of the content originally, this book is your chance to discover what you missed.

The ebook is being freely offered but we hope you will consider making a small donation. All proceeds benefit the OnRamp Scholarship fund maintained by The DevOps Collective, which is the organization behind the PowerShell Summit. The authors have freely contributed their content. Nobody is receiving any financial compensation. The content in this book may vary slightly from the originally published articles. The original content has been edited for consistency, grammar and spelling. Although be aware that not all authors are from North America!

Cover art graciously contributed by Jennifer Kanakos.

A Note On Code Listings

Due to page size limitations you may see some code samples in this book where the line ends in a backslash. This is Leanpub's line continuation character and is inserted when the line is too long to fit the width of the code sample. We've tried to adjust code as much as possible to reduce these breaks, but in some cases we felt it was easier to leave them in. Extended code samples are included in an extras zip file which you should have obtained when getting this book. All code is offered **as is** with no warranties or guarantees.

No code samples or snippets should be considered ready for production. They are for educational purposes only.

Questions and Comments

We are foregoing the usual book forum that can be setup with a Leanpub book. If you have questions, or comments about any of the content in this book, feel free to reach out to the individual author on Twitter. Check the Contributors page at the end of the book for a list of online links. You are also encouraged to use the free forums at PowerShell.org for all your other PowerShell questions or problems.

About OnRamp

OnRamp is designed for entry-level technology professionals who have completed foundational certifications such as CompTIA A+ and Cisco IT Essentials. PowerShell + DevOps Global Summit's[1] OnRamp track is a distinct ticket type that includes admission to a separate track of content designed for entry-level technology professionals. Less a conference and more of a guided, hands-on class, OnRamp is taught by some the industry's leading PowerShell instructors. It's more than just an introduction to PowerShell as a technology; OnRamp is also an introduction to the PowerShell community and ecosystem. By blending classroom time with time in Summit's general sessions, keynotes, and social events, OnRamp attendees can supercharge their entry into the broader world of DevOps and IT automation.

No prior PowerShell experience is required, although some basic knowledge of server administration will be useful. A number of full-ride scholarships designed to help bring new and diverse young professionals into the community and field are offered by the DevOps Collective. Your donation for this book goes towards this scholarship fund.

Learn more about the OnRamp Scholarship at https://events.devopscollective.org/OnRamp/Scholarship/.

[1] https://events.devopscollective.org/

About The DevOps Collective

Originally formed in 2012, The DevOps Collective, Inc.[2], is a US 501 C(3) nonprofit dedicated to education and community in the DevOps field. It specializes in technologies built around the PowerShell Language Specification (PLS), covering all platforms and operating systems. Aside from the PowerShell Summit[3] and the PowerShell.org website, it offers a number of free ebooks, free webinars, and more. Most PowerShell Summit sessions are recorded and offered for free on YouTube[4]. The organization also provide assistance to local user groups and enthusiasts who want to hold their own local events, such as a PowerShell Saturday or DevOps Day.

[2] https://devopscollective.org
[3] http://powershellsummit.org
[4] https://www.youtube.com/user/powershellorg

Foreword

> ***"PowerShell 7 is an open-source, cross-platform, automation and configuration tool enabling businesses to achieve greater agility in cloud, hybrid-cloud, and on-premises through reliable and repeatable expert automation."***

This sentence would not exist if not for you. Your shared discoveries and contributions are the foundation of PowerShell's success. I'm grateful that my friend, and top industry expert Jeff Hicks gathered a merry band of Super-Star PowerShell professionals together to share their PowerShell experiences. Let them show you how to flip today's challenges into tomorrow's success with PowerShell 7.

PowerShell loves technology. Open this book to explore the vast management and control you can automate with your existing technology investments. Is scaling service in the cloud part on your To-Do list? How about on-premises server management? Or do you have both in the form of a hybrid-cloud? Explore what these thought-leaders, experts and enthusiasts are thinking about, how they go about solving their daily challenges, and what failures impacted them along the way to success.

When you're finished with this book, help spread the word. Join with these authors and experts in helping everyone be successful with PowerShell. Help your friends and co-workers understand the benefits of adopting PowerShell 7 now!

Jason Helmick,

Program Manager | PowerShell Team

PowerShell 7 Experimental Features

by Dave Carroll

Experimental Features Defined

After becoming open-source software, the PowerShell community requested a mechanism for users to try out new features and provide early feedback to feature developers. This discussion took place in PowerShell RFC0029[5] which was finalized and implemented in PowerShell Core 6.1. New features that are not production ready are deemed experimental in nature. Users can choose to opt-in for an experimental feature on an individual basis. Administrators can choose to opt-in at the system level.

Please note that user configuration will take precedence over system configuration.

Using the built-in support for experimental features, developers can roll out an alternate command or a parameter to their modules. Experimental features are not limited to the PowerShell engine itself.

Experimental Feature Commands

Commands to discover, enable, and disable experimental features are provided to the user.

Get-ExperimentalFeature

The command `Get-ExperimentalFeature` will display a list of discovered experimental features. These features can come from the PowerShell engine itself or from modules. You can see where the experimental feature is defined by looking at the Source column in the command's output. Also, features can be specific to an operating system as the following illustrates. Notice the `PSUnixFileStat` feature in the Linux output.

[5] https://github.com/PowerShell/PowerShell-RFC/blob/master/5-Final/RFC0029-Support-Experimental-Features.md

Experimental Features on Windows

```
PS 7.0 > Get-ExperimentalFeature | Format-Table -Property Name,Enabled,Source -AutoSize

Name                                                  Enabled Source
----                                                  ------- ------
PSCommandNotFoundSuggestion                             False PSEngine
PSImplicitRemotingBatching                              False PSEngine
PSNullConditionalOperators                              False PSEngine
Microsoft.PowerShell.Utility.PSManageBreakpointsInRunspace   False C:\program files\powershell\7\Modules\Micr
PSDesiredStateConfiguration.InvokeDscResource           False C:\program files\powershell\7\Modules\PSDe
```

Get-ExperimentalFeature on Windows

Experimental Features on Linux

```
PS 7.0 > Get-ExperimentalFeature | Format-Table -Property Name,Enabled,Source -AutoSize

Name                                                  Enabled Source
----                                                  ------- ------
PSCommandNotFoundSuggestion                             False PSEngine
PSImplicitRemotingBatching                              False PSEngine
PSNullConditionalOperators                              False PSEngine
PSUnixFileStat                                          False PSEngine
Microsoft.PowerShell.Utility.PSManageBreakpointsInRunspace   False /opt/microsoft/powershell/7/Modules/Mic
PSDesiredStateConfiguration.InvokeDscResource           False /opt/microsoft/powershell/7/Modules/PSD
```

Get-ExperimentalFeature on Linux

Experimental feature discovery targets the paths in `$env:PSModulePath`.

Enable-ExperimentalFeature

The `Enable-ExperimentalFeature` command turns on one or more experimental features for the current user or all users. Enabling a feature will add it to an array in the `ExperimentalFeatures` key in the Powershell configuration file, `powershell.config.json`. If you do not specify a `Scope`, it will default to `CurrentUser`.

For Windows, the user configuration file will be saved in the `$HOME\Documents\PowerShell` folder. For Linux, the user configuration file will be saved in the `$HOME\config\powershell` folder.

On my system, I re-target my Documents folder to a separate volume. The PowerShell configuration file is saved there and is not in the `$HOME` hierarchy.

You can turn on all experimental features in one line. In the following example, I've added a sanity check by getting the content of the configuration file.

```
PS 7.0 > Get-ExperimentalFeature | Enable-ExperimentalFeature -Scope AllUsers
WARNING: Enabling and disabling experimental features do not take effect until next start of PowerShell.
PS 7.0 > Get-Content -Path $PSHOME\powershell.config.json | ConvertFrom-Json | Format-List

WindowsPowerShellCompatibilityModuleDenyList : {PSScheduledJob, BestPractices, UpdateServices}
Microsoft.PowerShell:ExecutionPolicy         : RemoteSigned
ExperimentalFeatures                         : {PSCommandNotFoundSuggestion, PSImplicitRemotingBatching, PSNullConditionalOperators,
                                               DemoExperimentalFeatures.ExperimentalFunction…}


PS 7.0 >
```

Enable-ExperimentalFeature All on Windows

> **Restart Sessions**
>
> Take note of the warning message that serves as a reminder to restart the PowerShell session. In fact, I believe you will need to close all console sessions (those of the same version and platform, that is) before the change will take effect. Don't forget the stop the terminal in Visual Studio Code.

Disable-ExperimentalFeature

The `Disable-ExperimentalFeature` command turns off the experimental feature. As with enabling, when you disable one or more features, you must close all PowerShell sessions and start a new session. Disabling the feature removes its entry from the enabled feature list in the appropriate configuration file. The ExperimentalFeatures key will remain even if you disable all experimental features.

PSUnixFileStat In Action

Now that we've talked about Experimental Features and how to enable/disable them, let's take one out for a spin. One of the features that Linux admins would probably appreciate is the `PSUnixFileStat` feature. Let's get some information about it.

```
PS 7.0 > Get-ExperimentalFeature -Name PSUnixFileStat | Format-List

Name        : PSUnixFileStat
Enabled     : True
Source      : PSEngine
Description : Provide unix permission information for files and directories
```

PSUnixFileStat Information

The default output for `Get-ChildItem` looks the same on Windows or Linux.

```
PS 7.0 > Get-ChildItem


    Directory: /home/wsladmin

Mode                 LastWriteTime         Length Name
----                 -------------         ------ ----
d----           7/14/2019 10:24 PM                go
-----           10/9/2019  1:01 AM             80 test.txt

PS 7.0 >
```

Get-ChildItem Before Enabling PSUnixFileStat

The output doesn't help the Linux admin with permissions. Let's enable the feature and correct that.

```
PS 7.0 > Get-ExperimentalFeature -Name PSUnixFileStat | Enable-ExperimentalFeature -Scope AllUsers
WARNING: Enabling and disabling experimental features do not take effect until next start of PowerShell.
PS 7.0 >
```

Enable-ExperimentalFeature PSUnixFileStat on Linux

I close all PowerShell sessions in my WSL instance and start a new one. And now to see the difference.

```
PS 7.0 > Get-ChildItem


    Directory: /home/wsladmin

UnixMode   User             Group                LastWriteTime         Size Name
--------   ----             -----                -------------         ---- ----
drwxrwxrwx wsladmin         wsladmin           7/14/2019 22:24          512 go
-rw-rw-rw- wsladmin         wsladmin           10/9/2019 01:01           80 test.txt
```

Get-ChildItem After Enabling PSUnixFileStat

Those new UnixMode entries look much more useful (and are super cool).

This is PowerShell on Linux!

PSCommandNotFoundSuggestion In Action

Let's take a quick look at another experimental feature.

```
PS 7.0 > get
get: The term 'get' is not recognized as the name of a cmdlet, function, script file, or operable program.
Check the spelling of the name, or if a path was included, verify that the path is correct and try again.

Suggestion [4,General]: The most similar commands are: set, gem, git, wget, del, ft, gal, gbp, gc, gci.
PS 7.0 >
```

PSCommandNotFoundSuggestion Sample Output

I meant to type *git*, not *get*. With the PSCommandNotFoundSuggestion experimental feature enabled, PowerShell can suggest commands when we have a typo or just space out.

Thanks, PowerShell!

Adding Support for an Experimental Feature to Your Module

As I mentioned at the beginning of this article, experimental features are not limited to the PowerShell engine. In fact, there are a couple delivered with PowerShell 7 within the modules `Microsoft.PowerShell.Utility` and `PSDesiredStateConfiguration`. I wanted to provide the community a working demo of experimental features, but I couldn't find any online.

My google-foo is strong, but it either failed me this time or there are no current examples in the wild.

I wrote a very simple demo module that contains experimental features. You can find it at the bottom of this article.

```
PS 7.0 > Get-ExperimentalFeature -Name DemoExperimentalFeatures.ExperimentalFunction,DemoExperimentalFeatures.ExperimentalParameter | Format-List

Name        : DemoExperimentalFeatures.ExperimentalFunction
Enabled     : False
Source      : C:\Program Files\PowerShell\Modules\DemoExperimentalFeatures\DemoExperimentalFeatures.psd1
Description : Demo of Experimental Functions

Name        : DemoExperimentalFeatures.ExperimentalParameter
Enabled     : False
Source      : C:\Program Files\PowerShell\Modules\DemoExperimentalFeatures\DemoExperimentalFeatures.psd1
Description : Demo of Experimental Parameter
```

Experimental Features Demo Module

Module Manifest

In the Module Experimental Feature[6] section of RFC0029, I found where experimental feature support can be added to a module manifest. In the `PrivateData.PSData` section, there is a new ExperimentalFeatures entry which allows an array of hashtables with Name and Description. This metadata has been incorporated into the necessary components to update the `PSModuleInfo` object.

[6] https://github.com/PowerShell/PowerShell-RFC/blob/master/5-Final/RFC0029-Support-Experimental-Features.md#module-experimental-feature

Feature Naming

When I was creating the demo module, I originally created a feature name like *PSDemoFeature*. I quickly discovered this was not the correct naming scheme for experimental features. It became evident when I tested my demo module manifest.

```
PS> Test-ModuleManifest 'C:\Program Files\PowerShell\Modules\Demo
ExperimentalFeatures\DemoExperimentalFeatures.psd1'
Test-ModuleManifest: One or more invalid experimental feature names found:
PSDemoExpFeature. A module experimental feature name should follow this
convention: 'ModuleName.FeatureName'.
```

Be sure to use the proper naming scheme for your experimental features. The name must be in the format of `ModuleName.FeatureName`.

The name of PowerShell engine experimental features is *PS*DescriptiveText. Once I realized this, I removed the *PS* from my feature names to reduce any confusion.

Experimental Attribute

The about_Experimental_Feature[7] documentation goes into detail on how to use the new `Experimental` attribute.

```
[Experimental(NameOfExperimentalFeature, ExperimentAction)]
```

This attribute can be used for the function or any parameter. The `ExperimentAction` is an `enum` with values of `Hide` or `Show`.

- `Show` will allow the experimental feature to be used when it's enabled.
- `Hide` will prohibit the experimental feature to be used when it's enabled.

They can be used to provide mutual exclusivity between different versions of a command or parameter.

Additional Information

Refer to the about_Experimental_Feature (referenced above) documentation for examples of C# and how to check if an experimental feature is enabled. The latter would be necessary when you don't need mutual exclusivity and when writing Pester tests for your code.

Demo Module with Experimental Features (Mutually Exclusive)

[7] https://docs.microsoft.com/en-us/powershell/module/microsoft.powershell.core/about/about_experimental_features?view=powershell-7

```
#DemoExperimentalFeatures.psd1

@{

    RootModule           = 'DemoExperimentalFeatures.psm1'

    ModuleVersion        = '0.7.0'

    CompatiblePSEditions = 'Core'
    GUID                 = 'a007643e-c876-4806-b6cb-367963716e98'
    Author               = 'Dave'
    CompanyName          = 'thedavecarroll'
    Copyright            = '(c) Dave. All rights reserved.'

    PowerShellVersion    = '7.0'

    FunctionsToExport    = 'Show-HelloWorld', 'Get-LoremIpsum'

    PrivateData          = @{
        PSData = @{

            ExperimentalFeatures = @(
                @{
                    Name        = 'DemoExperimentalFeatures.ExperimentalFunction'
                    Description = 'Demo of Experimental Functions'
                },
                @{
                    Name        = 'DemoExperimentalFeatures.ExperimentalParameter'
                    Description = 'Demo of Experimental Parameter'
                }
            )
        }
    }

}
```

```
#DemoExperimentalFeatures.psm1

function Show-HelloWorld {
    [Experimental("DemoExperimentalFeatures.ExperimentalFunction", "Show")]
    [CmdletBinding()]
    param()

    'PowerShell 7 is here!' | Write-Host -ForegroundColor Yellow
}

function Show-HelloWorld {
    [Experimental("DemoExperimentalFeatures.ExperimentalFunction", "Hide")]
    [CmdletBinding()]
    param()

    'PowerShell 7 is shipping soon!' | Write-Host -ForegroundColor Green
}

function Get-LoremIpsum {
    [CmdletBinding()]
    param(
        [Experimental("DemoExperimentalFeatures.ExperimentalParameter", "Show")]
        [switch]$Show,

        [Experimental("DemoExperimentalFeatures.ExperimentalParameter", "Hide")]
        [switch]$Display
    )

    $LoremIpsum = @"
Lorem ipsum dolor sit amet, consectetur adipiscing elit. Sed varius mi erat,
in laoreet nibh eleifend eget. Phasellus odio diam, tincidunt rhoncus massa
in, feugiat iaculis mauris. Nulla ornare enim et semper tincidunt. Maecenas
ac tempor quam, in scelerisque lorem. Duis hendrerit urna sapien, ut
pellentesque odio placerat finibus. Mauris molestie nulla ac vestibulum
aliquet. Quisque ut lacus quis lorem venenatis elementum. Morbi hendrerit
odio at nibh faucibus finibus. Vivamus porta elit libero, et porttitor leo
malesuada et. Suspendisse luctus erat sed lectus ultricies laoreet.
"@

    $fox = 'The quick brown fox jumps over the lazy dog'

    if ($Display) {
        Write-Output $LoremIpsum
```

```
    }

    if ($Show) {
        Write-Output $fox
    }
}
```

This code is available in the Extras zip file.

Summary

I believe module developers will start delivering experimental features as they migrate to PowerShell 7, especially if their modules support mission critical automation and processes. Thank you for taking the time to read this article and for being part of the PowerShell community. You are the reason we do what we do!

If you haven't already, begin your journey with PowerShell 7 now!

This article originally appeared online at https://powershell.anovelidea.org/powershell/ps7now-experimental-features/

Contributors

The following people were kind enough to contribute their time and energy towards this project. Not only did they create and publish original content on their respective blogs or web sites, but they also took the extra step to provide their content for this book.

Adam Bertram

Adam Bertram is a 20+ year veteran of IT and an experienced online business professional. He's a consultant, Microsoft MVP, blogger, trainer, published author and content marketer for multiple technology companies.

Dan Franciscus

Dan Franciscus is a veteran IT professional specializing in PowerShell, Active Directory, Windows Server, and generally automating stuff. His current role focuses on end-user digital experience using data science techniques.

Dave Carroll

Dave Carroll has spent more than two decades in IT in various capacities with significant focus on systems administration. For most of this time, he developed automation through multiple scripting languages, with a strong focus on PowerShell over the last 10 years. Currently, he is an Identity and Access Management Application Developer for a top university.

You can find more of Dave's content online on Github at https://github.com/thedavecarroll.

Jeff Hicks

Jeffery Hicks is an IT veteran with almost 30 years of experience, much of it spent as an IT infrastructure consultant specializing in Microsoft server technologies with an emphasis on automation and efficiency. He is a multi-year recipient of the Microsoft MVP Award. He works today as an independent author, teacher and consultant. Jeff has taught and presented on PowerShell and the benefits of automation to IT Pros worldwide. He has authored and co-authored a number of books, writes for numerous online sites, a Pluralsight author, and a frequent speaker at technology conferences and user groups.

Jonathan Medd

Jonathan is a Cloud Automation Engineer at Atos. A PowerShell MVP between 2010 - 2019 and VMware vExpert between 2011 and 2020, he is the co-author of VMware vSphere PowerCLI Reference 1st and 2nd editions and co-organiser of both PSDayUK, a PowerShell one day conference in the UK, and PowerShell Southampton.

Josh Duffney

Josh Duffney is a DevOps engineer with 10 years of systems administration and engineering experience. Josh is a Pluralsight author of several courses on the topic of automation and infrastructure development and blogs at duffney.io. Josh enjoys being an active member of the PowerShell and DevOps community where he gets to share his knowledge but more importantly enjoys learning from others in the industry. Outside of his professional work Josh is a practitioner of digital minimalism who strives to find balance in a digital world. Josh also spends his time weight lifting and training in the art of Brazilian jiu-jitsu.

Josh King

Josh King is a Microsoft MVP and Systems Administrator at Tribe, an IT services organization in New Zealand. Josh predominantly works within Windows and VMware environments. He is also a contributor at TechSnips[8], an e-learning tech screencast platform. Josh has a passion for automation and his primary tool of choice is PowerShell. He also has an unhealthy obsession with toast notifications and how they can be leveraged for script to operator communication.

Mike Kanakos

Mike Kanakos is a 20 year IT infrastructure pro currently working as a Senior Systems Engineer and Identity Access Management specialist. Mike is also the co-leader of the Research Triangle PowerShell users group[9], community lead for PowerShell.org, active blogger and public speaker. He focuses on teaching PowerShell fundamentals and sharing tools with the community to make a sysadmin's life easier.

Prateek Singh

Prateek Singh is an Infrastructure Developer working at LinkedIn[10], an avid PowerShell blogger, and a community contributor. In 2017 and 2018, his blog RidiCurious.com as among the "Top 50

[8] https://techsnips.io/
[9] https://rtpsug.com/
[10] https://www.linkedin.com/in/prateeksingh1590/

PowerShell blogs in the world". Find his work open-sourced at GitHub[11], and at Leanpub[12].

Thomas Lee

PowerShell/Lync Geek, Grateful Dead/Jerry Live Recording Enthusiast and living in the UK.

Tommy Maynard

Tommy Maynard is a Senior Systems Administrator with a passion for PowerShell. He has nearly 20 years of experience in Information Technology, and PowerShell has so far, been the most rewarding part of his overall career. Luckily for him, PowerShell works alongside the technologies he has long supported. It often even works with the new ones, too. His goal is to help educate and inspire people that work in his industry to embrace PowerShell, and in general, automation. Tommy lives in Tucson, Arizona with his wife Jenn and two kids, Carson and Arianna.

#PS7Now Contributors Online

Author	Twitter	Blog
Josh King	https://twitter.com/WindosNZ	https://toastit.dev/
Josh Duffney	https://twitter.com/joshduffney	http://duffney.io/
Adam Bertram	https://twitter.com/adbertram	https://adamtheautomator.com/
Mike Kanakos	https://twitter.com/MikeKanakos	https://www.networkadm.in/
Jonathan Medd	https://twitter.com/jonathanmedd	https://www.jonathanmedd.net/
Thomas Lee	https://twitter.com/doctordns	https://tfl09.blogspot.com/
Prateek Singh	https://twitter.com/singhprateik	https://ridicurious.com
Dave Carroll	https://twitter.com/thedavecarroll	https://powershell.anovelidea.org/
Dan Franciscus	https://twitter.com/dan_franciscus	https://winsysblog.com/
Jeff Hicks	https://twitter.com/jeffhicks	https://jdhitsolutions.com/
Tommy Maynard	https://twitter.com/thetommymaynard	https://tommymaynard.com/

[11] https://github.com/PrateekKumarSingh
[12] https://leanpub.com/b/books