

# DEVELOPING AND TESTING CROSS PLATFORM COMPATIBLE MODULES WITH POWERSHELL CORE



MARK WRAGG

DevOps Engineer, XE.com

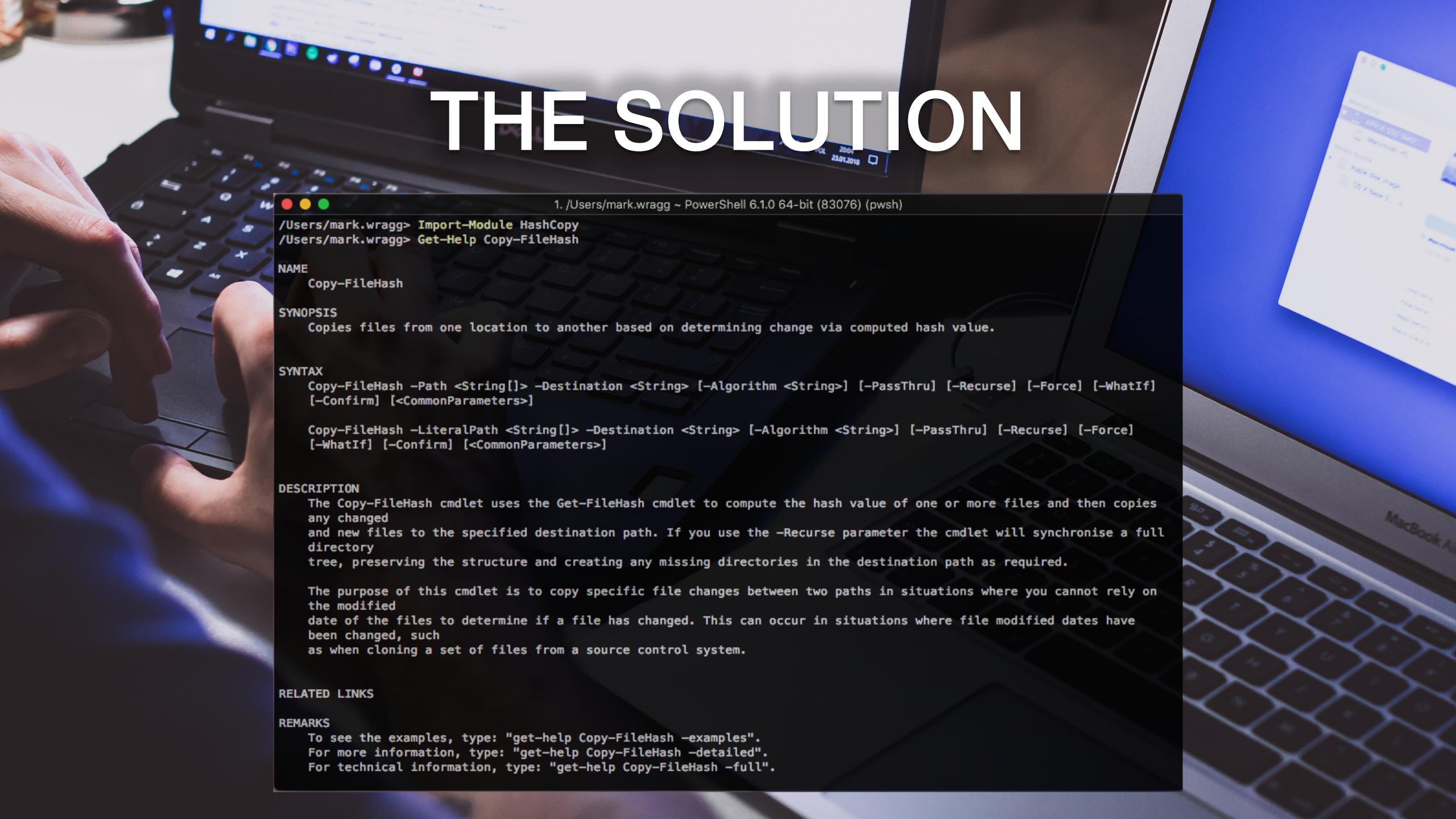
@markwragg

<https://wragg.io>

# THE PROBLEM

- Automate deployment of static content files
- Support two deployment scenarios:
  - Dev: Deploy changes during regular monthly releases
  - Ops: deploy between releases and without downtime
- Allow only modified files to be deployed when required.

# THE SOLUTION



```
1. /Users/mark.wragg ~ PowerShell 6.1.0 64-bit (83076) (pwsh)
/Users/mark.wragg> Import-Module HashCopy
/Users/mark.wragg> Get-Help Copy-FileHash

NAME
    Copy-FileHash

SYNOPSIS
    Copies files from one location to another based on determining change via computed hash value.

SYNTAX
    Copy-FileHash -Path <String[]> -Destination <String> [-Algorithm <String>] [-PassThru] [-Recurse] [-Force] [-WhatIf]
    [-Confirm] [<CommonParameters>]

    Copy-FileHash -LiteralPath <String[]> -Destination <String> [-Algorithm <String>] [-PassThru] [-Recurse] [-Force]
    [-WhatIf] [-Confirm] [<CommonParameters>]

DESCRIPTION
    The Copy-FileHash cmdlet uses the Get-FileHash cmdlet to compute the hash value of one or more files and then copies
    any changed
    and new files to the specified destination path. If you use the -Recurse parameter the cmdlet will synchronise a full
    directory
    tree, preserving the structure and creating any missing directories in the destination path as required.

    The purpose of this cmdlet is to copy specific file changes between two paths in situations where you cannot rely on
    the modified
    date of the files to determine if a file has changed. This can occur in situations where file modified dates have
    been changed, such
    as when cloning a set of files from a source control system.

RELATED LINKS

REMARKS
    To see the examples, type: "get-help Copy-FileHash -examples".
    For more information, type: "get-help Copy-FileHash -detailed".
    For technical information, type: "get-help Copy-FileHash -full".
```

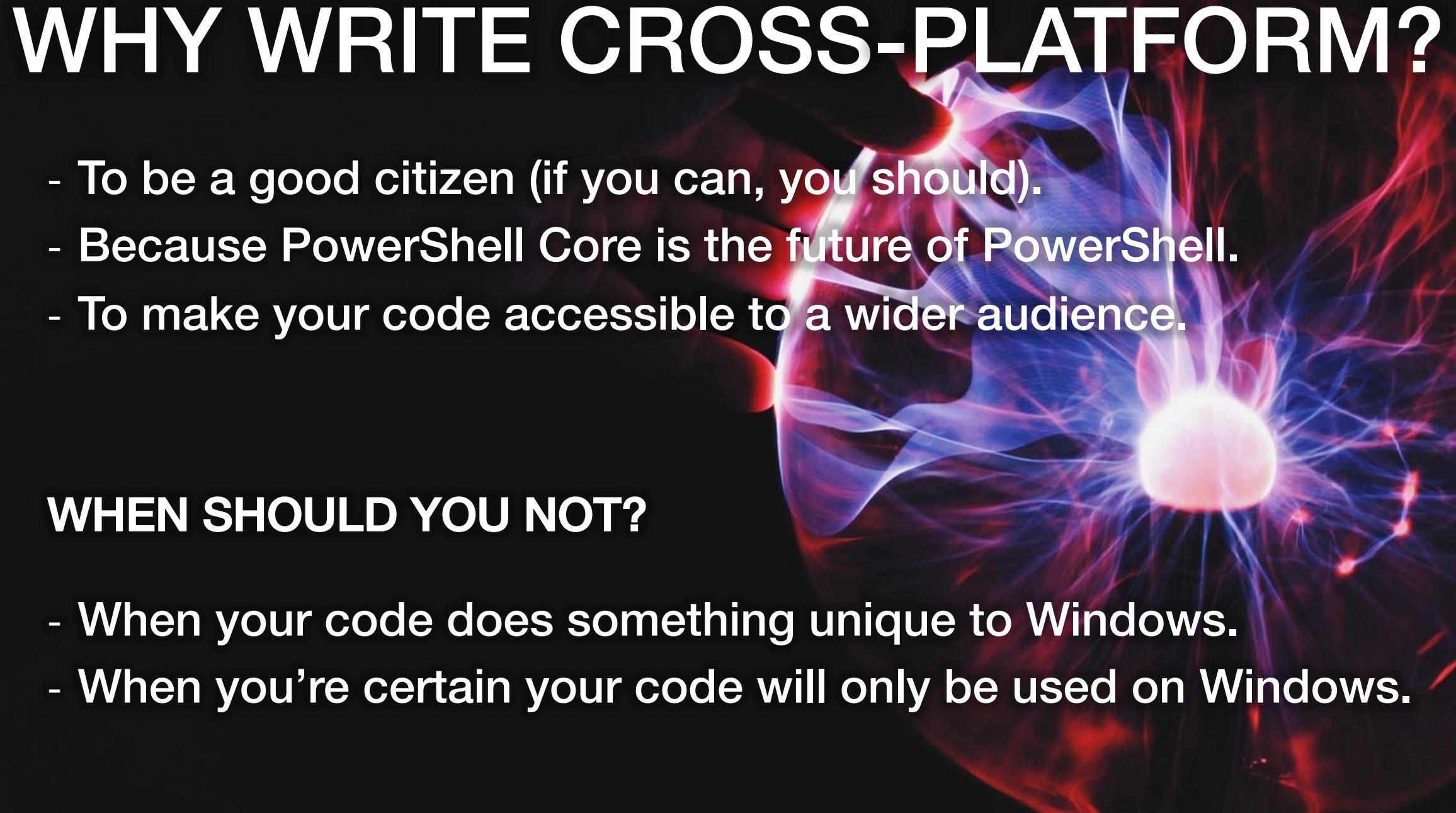
# POWERSHELL CORE

- New, fast
- Open-source, cross-platform
- Not all cmdlets available / modules compatible

# THINGS TO CONSIDER

- Cmdlet availability / avoid aliases.
- File paths: use forward slashes, reference system paths via environment variables, use -path cmdlets.
- Encode as UTF8.

# WHY WRITE CROSS-PLATFORM?



- To be a good citizen (if you can, you should).
- Because PowerShell Core is the future of PowerShell.
- To make your code accessible to a wider audience.

## WHEN SHOULD YOU NOT?

- When your code does something unique to Windows.
- When you're certain your code will only be used on Windows.

# TESTING YOUR CODE

- Pester is the PowerShell testing framework.
- Beware the same cross-platform considerations apply to your test code (e.g. be wary of how you handle paths).
- A thorough set of tests is the best way to catch compatibility issues.

# DEMO: APPVEYOR

A photograph of a person's hands holding a lit sparkler. The sparks are bright yellow and white, contrasting sharply with the deep blue and black tones of the background. The person is wearing a dark long-sleeved shirt.

- Leveraging the free Continuous Integration service AppVeyor to perform per-commit Pester testing of code on both a Linux and Windows VM.

# SUMMARY

- Review your existing code for opportunities to make it cross-platform (or confirm if it already is!).
- You can ensure your code remains cross-platform compatible via automated testing.
- Tag your PS Core compatible code as such in the PS Gallery.
- Links:
  - <https://www.powershellgallery.com/packages/HashCopy/>
  - <https://github.com/markwragg/PowerShell-HashCopy>
  - <https://www.appveyor.com/>

Twitter: @markwragg  
Blog: <https://wragg.io>