Unpractical Powershell

Who Needs This book?

When I first started as an IT professional 20+ years ago I loved the gui (graphical user interface). It was safe, it was friendly, like a fluffy bunny on a warm spring day. Powershell unpractical? Of course not. So, before I decided to write this book I sat down and started to make a list of all the things I do with Powershell or how Powershell makes my life easier. After all, if Powershell isn’t helping to satisfy your internal “lazy bastard,” as my friend Harry calls his need to automate things, what’s it doing for you anyway?

I started to think about some of my colleagues and when I start spouting Powershell solutions I can hear them saying, “here he goes again, going on and on about something I’m just not ready to invest my time and effort in, after all Microsoft changes these things all the time?” Right? Wrong. Powershell *is* the new Microsoft as far as any IT Professional is concerned. Microsoft has made a massive investment in the Powershell language. If you want to keep your job as an IT professional learn it now. For any pr

What IDE to Use?

Just like any good artist you’ll need a pallet to work with. Before you start writing your Powershell you need to make sure you have an Integrated Development Environment you are comfortable with. I’m not a Microsoft employee and have no biases. Its simply up to you. However, throughout this book we will be using Microsoft Visual Studio Code. I feel like you can’t beet the IDE from the same company that developed the runtime. So, without further ado. Let’s get started.

One of the common things you may find yourself doing is calling an executable from a Powershell. Many times there are executables you might want to whittle down command line arguments for an executable that you need to run often.

Our first example is actually going to be dealing with git. Git executable. I won’t go into detail about what Git is but that it is a source control for the software you write and that if you write any software you should be using it.

Functions and Parameters

Try/Catch/Trap Error Handling

Running commands with parameters from Powershell

Backup from Powershell

Accessing WMI/CIM objects via Powershell

Containers from Powershell

Generating User Interfaces with Powershell

Working with the Web

Invoke-WebRequest

function test-url ($url) {

$HTTP\_Request = [System.Net.WebRequest]::Create($url)

# We then get a response from the site.

$HTTP\_Response = $HTTP\_Request.GetResponse()

# We then get the HTTP code as an integer.

$HTTP\_Status = [int]$HTTP\_Response.StatusCode

If ($HTTP\_Status -eq 200) {

Write-Host "Site is OK!"

return $true

}

Else {

Write-Host "The Site may be down, please check!"

return $false

}

# Finally, we clean up the http request by closing it.

$HTTP\_Response.Close()

}