Unpractical Powershell

## Why Read 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 under the sun. You pressed the button and the computer did something. Pressing buttons and watching status bars…Pressing buttons and watching status bars…Then after pressing the buttons and watching the status bars hundreds or even thousands of times I started to wonder. I started to wonder to myself, “Is this all there is with IT, pressing buttons and watching status bars?” I then started to explore the ways I could make my job easier and sort of offload some of the repetitive tasks. Throughout school, I’ve had a fair amount of math and programming. I understood algorithms and loops. So, like any good IT professional of the time, I started writing VBScripts. Oh yeah, like He-Man use to say, holding up his sword “I have the power!” VBScript was awesome. I could reference libraries for reuse just like writing Visual Basic programs, but in a script with no compile and packaging.

A few years have passed and thousands of lines of VBScript under my belt. Since then VBScript has gotten a bad rap and for good reason. It served its purpose and now a new kid is on the block. Powershell.

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?

Powershell unpractical? Of course not. But this book isn’t for the practicality of Powershell. There are plenty of those. This book won’t teach you for loops and variable declaration. There are enough of those, written by some really great people. This book is to express the elegance of design behind Powershell, to show the things us “lazy bastards” may be missing, or what may be considered “unpractical” because, quite frankly, you didn’t know Powershell could do that.

Things covered in this book may not be particular to your job function but the hope is that it helps and gives you a greater appreciation for the language of Powershell.

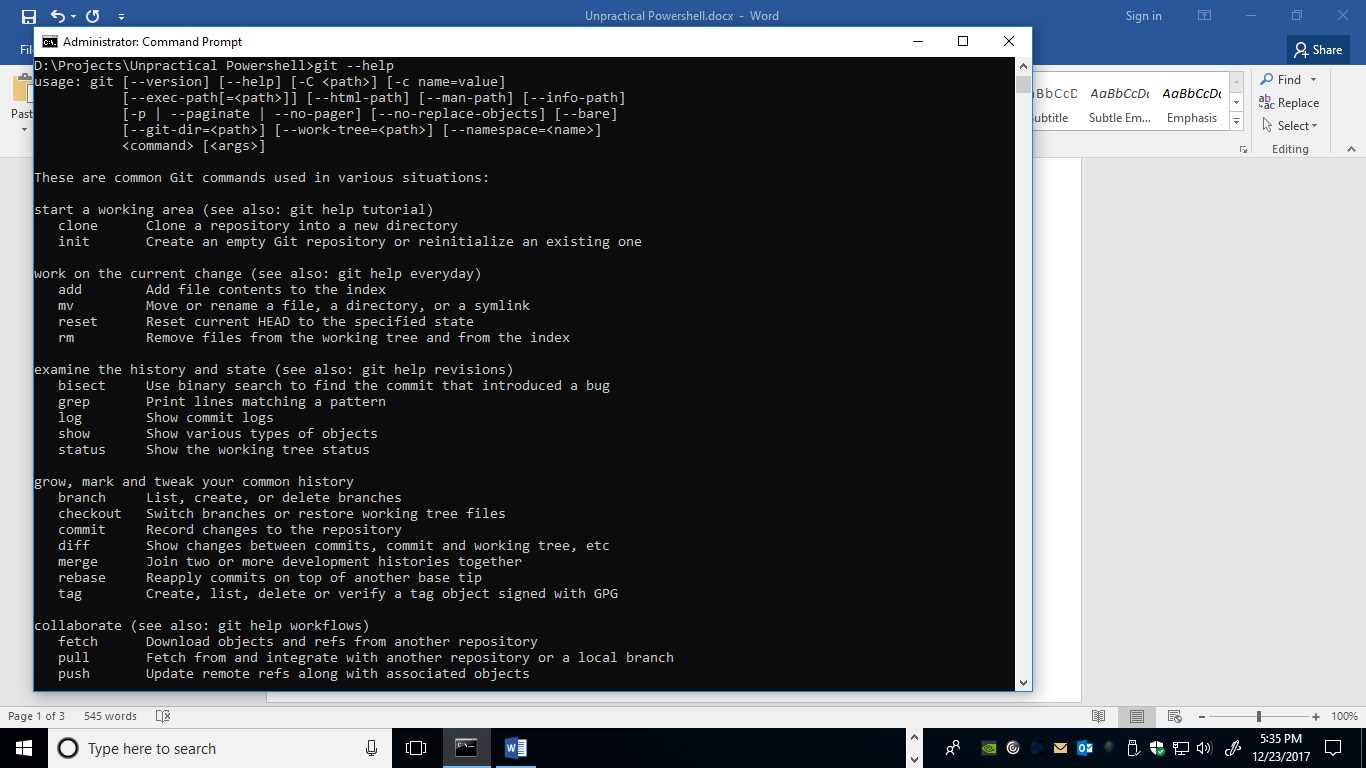
## Being an Artist

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. Visual Studio Code will run on Windows, Mac, and Linux. 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 that you need to run often.

Our first example is actually going to be dealing with Git, particularly the Git executable. I won’t go into detail what Git is, but that it is a source control for the software you write. If you write any software you should be using it, wheatear that is scripting or compiled code. Using Git is part of becoming a professional IT person. If that doesn’t appeal that’s ok you’ll still get the jest of what we are trying to do.

First, if you don’t have Git you can download your version at github.com. You need to make sure that when you run git - - help from a command line you get something similar to the below.



This ensures that Git is running from a command line prompt. You should also make sure the Git path is defined in your system path for this example. This ensures that running Git from any folder will provide the same results.

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()

}