Powered by Zerto

TOP 10 PowerShell tips and tricks

#1 VARIABLES EVERYWHERE

var var var var

Learning to use variables frequently in your scripts makes them easier to write, troubleshoot and reduces complexity. Variables should be used any time you need to use the same value multiple times or to store information for subsequent use. Amongst many use cases, variables can also be used to perform calculations, store imported data, results of cmdlets, sort and filter objects. Ensure you always give your variables descriptive names and make them unique to avoid confusion or accidentally overwriting data. Making frequent use of variables will set you on the fast track to becoming a PowerShell ninja.

```
1   $variable1 = 1
2   $variable2 = 2
3   $variable3 = $variable1 + $variable2
4   $variable3
```

#2 COMMENT, COMMENT, COMMENT

Using comments can make the difference between a script being a complete nightmare, that is impossible to troubleshoot vs being readable, concise and easy to troubleshoot. Every 3-10 lines of script should have a comment describing what the section of the script is doing and why. Complex elements should have their own comments to help you understand their function when reviewing or troubleshooting the script. You can also use comments to create sections of script and indexes to help you find your way around complex long scripts!

```
##########
# 1. Start of script
##########
# A comment goes a long way
sleep 10 # To making a script readable
```

#3 KEEP IT SIMPLE

When using PowerShell there are many ways to get the script to perform the action needed. This does however mean that there is lots of scope for creating scripts of extreme length with vast amounts of complexity, when the same feat could have been achieved with significantly less. A good script writer will always get a new section of a script working first, then boil it down to the smallest number of lines required to maintain simplicity and readability. If you look at a line of script and have no clue what it is doing, then it probably needs to be revised or an explanation added if there is no alternative. Failing to do this makes it extremely hard to troubleshoot larger scripts. So always try to keep this in mind as you build out your scripts.

```
# There is always a hard way
$GetTime = Get-Date | select -ExpandProperty TimeOfDay | select Hours,Minutes,Seconds
$Hours = $GetTime.Hours
$Minutes = $GetTime.Minutes
$Seconds = $GetTime.Seconds
$Time = "$Hours" + "-" + "$Minutes" + "-" + "$Seconds"
# And an easy way
$Time = Get-Date -Format "H-mm-ss"
# To get what you want in PowerShell
```

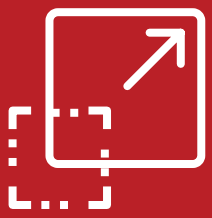
#4 TESTING

PowerShell scripting is like building multiple lego sets, each with moving parts, that need to interact with each other when you've finished building them all. You can certainly build them all then at once and test it at the end, but this will often result in extremely complex and long winded troubleshooting. It is much more efficient to test each set as you go along, I.E each new section of script, then only test the whole script when finished. Temporarily assign variables to simulate the inputs each section is expecting and manually check the value of variables as you go along. Using a script editor like PowerShell ISE is also going to help you not only write your script quicker, with suggested cmdlets, it will allow you to easily perform the testing.

```
# Section 1, test this first
$PID
# Section 2, otherwise you will miss it
kill $PID
```

Powered by Zerto

#5 LOGGING EXCEPTIONS



Troubleshooting scripts is a common task in the life of a script writer. One vital tool in your arsenal is Start-Transcript and Stop-Transcript to log the actions of the script and to catch any exceptions it meets. Using transcripts ensures you can find out what went wrong without having to run the script again while also being able to retain historical errors. Create logical naming conventions for your transcript to ensure you maintain a history and not overwrite the log on every run. You can also write variables to the console for them to be included in the transcript. This makes it easier to troubleshoot the script by logging the data the script is working with or producing at each stage.

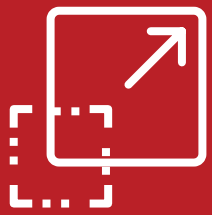
```
$DateTime = Get-Date -Format "MM.dd.yy H-mm-ss"
$Transcript = "c:\test\transcript - " + $DateTime + ".txt"
$Test = "Hello World"
Start-Transcript -Path $Transcript -NoClobber
$Test
Stop-Transcript
```

#6 TRY, CATCH & TRY AGAIN

Transcription logging is a certainly a good practice, but what happens if you don't capture the full error? A log is no good if it doesn't contain all the information needed. Using Try and Catch as a wraparound to your commands gives you the power of capturing critical errors in detail, setting variables based on outcomes and performing the same or different actions by nesting commands. This can be useful when interacting with remote systems or to try another action if the command fails. Another important use is to force a script to terminate or not, which can be handy if you don't want the script to continue if it errors or the exact opposite. Using try and catch wrappers will go a long way to making your scripts more robust and easier to troubleshoot.

```
Try
{
connect-viserver -Server $vCenterServer -User $Username -Password $Password
$vCenterAuthentication = "PASS"
}
Catch
{
$Error[0] | Format-List -Force
$vCenterAuthentication = "FAIL"
}
$vCenterAuthentication
```

#7 CSVS ARE YOUR BEST FRIEND



Managing the input and output of data in PowerShell is a common task. If you have a list of data to process, or data generated in PowerShell to export, then Import-CSV and Export-CSV is an extremely powerful way of getting useful information both in and out of your script. Create some CSVs and start playing with it now if you haven't used these commands before, as their use is only limited by your imagination. Need to perform an action against a list of VMs, hosts or IP addresses? Need to export the running processes or the result of a query? Need to log the information returned for reporting purposes? This is where Import-CSV and Export-CSV come into their own so there is no better way to learn other than to go start using them today.

```
$CSV = "c:\Test\ProcessCSVExport.csv"
$Processes = Get-Process
$Processes | Export-Csv $CSV -NoTypeInformation
$Import = Import-Csv $CSV
$Import | Format-Table
```

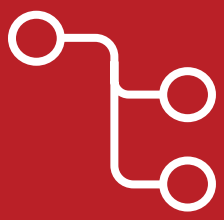
Powered by Zerto

#8 IF THIS... THEN THAT

Conditional IF statements are one of the most useful aspects of PowerShell when you need to change the action of a script depending on the result of a test, output from a query, line from a CSV or string in an array. In many scripts, you often need to perform a completely different set of actions depending on what a variable has been set to. I.E when testing if a file or folder exists or pinging a VM. An IF statement allows you to trigger certain actions, such as email notifications, if the test fails. You can also use IF statements to bypass sections of script and nest them within each other for more advanced use cases. A quick tip when creating large IF statements is to add a comment before and after the closing bracket explaining which statement it belongs to, as this aids with troubleshooting at scale.

```
$PingTest = Test-Connection -ComputerName localhost -Count 2 -Quiet
$FileTest = Test-Path "c:\test\"
if (($PingTest -eq $False) -or ($FileTest -eq $False))
{
write-host "Test Failed"
}
else
{
write-host "Tests Passed"
}
```

#9 FOR EACH DO THIS



The whole purpose of a script is to automate manual tasks. To do this scripts are often run multiple times to complete the task at hand, but this can still take time and can introduce new problems. If you have to repeat an action for 100 VMs, 200 users or 50 mailboxes then ForEach makes the process simple. To use ForEach you first assign a list of strings to variable, manually, from a cmdlet or even a CSV. ForEach will then allow you to run the same script block for each object to fully automate the task as many times as required in one simple script. Combining ForEach and IF statements allows you to build scripts that handle complex scenarios in a simple programmatic fashion. If you haven't ever used ForEach statements then now is a good time to start learning.

```
$List = "Host1","Host2","Host3","Host4"
$Count = 0
foreach ($_ in $List)
{
write-host $_
$Count ++
}
$Count
```

#10 SEARCH

Last but certainly not least, google is one of the best tools at your disposal when writing PowerShell scripts. It is a myth to think that even experienced PowerShell scripters can sit and write a script from scratch. The best PowerShell scripters are often the most proficient at finding examples of existing scripts and tweaking them to work for the task at hand. It's very rare that somebody hasn't already scripted the same thing you that you are building, or at least the same thing but with a completely different use case. Don't be afraid to copy, paste and tweak multiple examples to fit your use case as you build out your own library of scripts to reference. You don't have to know everything from memory to be a PowerShell expert. You just need to know where to go, how to get it working and most importantly, have fun learning!

Powered by Zerto