Powershell - Reference

By John Bentley

Basics

Setup

Start Icons

Turn on administrative tools.

- Windows Start > [Top Right hand Corner] > Settings > Tiles > Show Administrative Tools: On.

To setup Powershell ISE to the Start Screen:

- Control Panel\System and Security\Administrative Tools
- " Windows PowerShell ISE (x86)" > Right Click > Pin to Start [We are using the 32 bit version]
- " Windows PowerShell (x86)" > Right Click > Pin to Start [We are using the 32 bit version]

We use the 32 bit version of Powershell because that's the one that launches from ISE > File > Start Powershell.exe regardless of whether the 64 bit version of the ISE is used or the 32 Bit version of ISE is used.

To change the font of the PowerShell

- "C:\ProgramData\Microsoft\Windows\Start Menu\Programs\Administrative Tools\Windows PowerShell (x86).lnk" > Right Click > Run as Administrator.
- Powershell Window Bar (up the top) > Right Click > Properties (not Defaults) > Font > Raster Fonts: 8 X 12

Lucinda Console Fonts don't seem to work.

Setup or Update Powershell Core (6.x)

To setup or update Powershell Core (6.x):

- Download and install "Windows (x64)" .msi from https://github.com/PowerShell/PowerShell
- The installer creates a shortcut in the Windows Start Menu.
    - By default the package is installed to $env:ProgramFiles\PowerShell\<version>
    - You can launch PowerShell via the Start Menu or $env:ProgramFiles\PowerShell\<version>\pwsh.exe

https://github.com/PowerShell/PowerShell
https://docs.microsoft.com/en-us/powershell/scripting/install/installing-powershell-core-on-windows?view=powershell-6

Setup Powershell

To setup Powershell to open at a particular directory, use a profile:

- [On Windows Start] > powershell
- Create a Powershell profile:
    - To display "Current user, Current Host" profile path for PS powershell:
      PS>$profile
    - Returns:
      C:\Users\John\Documents\PowerShell\Microsoft.PowerShell_profile.ps1
    - Create a blank "Current user, Current Host" profile
      PS> if (!(Test-Path $profile)) {new-item -ItemType file -Path $profile -Force}

- Add working directory to C:\Users\John\Documents\PowerShell\Microsoft.PowerShell_profile.ps1: (e.g.)
  cd C:\Users\John\Documents\Sda\Code\Windows\PowerShell

(Microsoft, 2014. Scripting with Windows PowerShell) "about_Profiles", https://technet.microsoft.com/en-us/library/hh847857.aspx (or Get-help about_Profiles)

Setup Powershell ISE

To code Powershell Scripts you can use any text editor but it's best to use Windows Powershell Integrated Scripting Environment (ISE).

To setup Powershell ISE to open at a particular directory, use a profile:

- [On Windows Start] > powershell_ISE
- Create a Powershell profile:
  - To display "Current user, Current Host" profile path for PS powershell_ISE:
    PS>$profile
  - Returns:
    C:\Users\John\Documents\PowerShell\Microsoft.PowerShellISE_profile.ps1
  - Create a blank "Current user, Current Host" profile
    PS> if (!(Test-Path $profile)) {new-item -ItemType file -Path $profile -Force}
  - Add working directory to C:\Users\John\Documents\PowerShell\Microsoft.PowerShellISE_profile.ps1 : (e.g.)
    cd C:\Users\John\Documents\Sda\Code\Windows\PowerShell

(Microsoft, 2014. Scripting with Windows PowerShell) "about_Profiles", https://technet.microsoft.com/en-us/library/hh847857.aspx (or Get-help about_Profiles)

Update powershell help:

- ISE (as Admin) > Help > Update Windows Powershell Help; or
- PS (as Admin) > update-help

Permissions

To run scripts use Set-ExecutionPolicy cmdlet to change the execution policy to AllSigned or RemoteSigned.

```
Set-ExecutionPolicy -Scope Process -ExecutionPolicy RemoteSigned
```

The RemoteSigned execution policy requires a digital signature on scripts that you download or get from other computers, but it does not require a digital signature on scripts that you write on your local computer.

-Scope <ExecutionPolicyScope>
-- Process: The execution policy affects only the current Windows PowerShell process.
-- LocalMachine: The execution policy affects all users of the computer.

Create Right Click Command

To install: save the following to a *.reg file and double click it. To use: in Windows Explorer select a folder, and press Shift + Right Click.

```
Windows Registry Editor Version 5.00
```

```
[HKEY_CLASSES_ROOT\Directory\shell\PSOpenHere]
@="Open powershell here"
"Icon"="C:\\Windows\\SysWow64\\WindowsPowerShell\\v1.0\\powershell.exe,0"
"Extended"=""

[HKEY_CLASSES_ROOT\Directory\shell\PSOpenHere\command]
@="C:\\Windows\\SysWOW64\\WindowsPowerShell\\v1.0\\powershell.exe -NoExit -Command Set-
Location -LiteralPath '%L'"
```

(Sheppard, 2015. Windows PowerShell command line syntax) Open PowerShell Window Here, http://ss64.com/ps/syntax-openhere.html
\\Atlas\Documents\Sda\Code\Windows\PowerShell\Libraries\CreateOpenPowerShellHere.reg

Run

Run PowerShell ISE

Once setup run Powershell ISE:

- By clicking on the Start icon "powershell_ise".
- From a command line> PowerShell_ISE
- Windows Search> ISE
- Windows Explorer > Address Bar > [Type]: "Powershell ise" (this opens the ISE at the location currently set in windows explorer).

(Microsoft, 2014. Scripting with Windows PowerShell) "Starting Windows PowerShell on Windows 8 and Windows", https://technet.microsoft.com/en-us/library/hh847889.aspx
Stackoverflow, How to start PowerShell from Windows Explorer?, Ashwin Nanjappa, http://stackoverflow.com/a/6599296

Run Powershell

Run Powershell:

- By clicking on the Start icon "Windows PowerShell".
- Windows Powershell ISE > File > Start PowerShell.exe (Ctrl + Shift + P)
- From a DOS command line> Powershell
- From Windows Key > Run: Powershell
- Windows Search> Powershell

(Microsoft, 2014. Scripting with Windows PowerShell) "Starting Windows PowerShell on Windows 8 and Windows", https://technet.microsoft.com/en-us/library/hh847889.aspx

Run Powershell Script

Run powershell script using one of the following techniques …

Via the right click menu: right click on *.ps1 script > Run with PowerShell.

Via a shortcut:

1. Right click on the script to create a shortcut.
2. Modify the shortcut's target field to read something like …

```
// Run using Windows Terminal (and Powershell 7x)
C:\Users\John\AppData\Local\Microsoft\WindowsApps\wt.exe "C:\Program
Files\PowerShell\7\pwsh.exe" -command "&
'C:\Users\John\Documents\Sda\Code\Windows\PowerShell\Apps\xStopProcesses\xStopProcesse
s.ps1'"

// Powershell 6
"C:\Program Files\PowerShell\6\pwsh.exe" -command "&
'C:\Users\John\Documents\Sda\Code\Windows\PowerShell\Apps\xStopProcesses\xStopProcesse
s.ps1'"
```

```
// Powershell 5 and bellow.
C:\Windows\System32\WindowsPowerShell\v1.0\powershell.exe -command "&
'C:\Users\John\Documents\Sda\Code\Windows\PowerShell\Apps\xStopProcesses\xStopProcesse
s.ps1' -MyArguments Something"
```

3. Double click the shortcut.
4. Optionally pin the shortcut to the start menu or taskbar and click from there.

https://stackoverflow.com/a/10137272/872154, "Stackoverflow, "Is there any way to make powershell script work by double clicking .ps1 file?", answer by David Brabant 2012-04-13.

Run From Powershell

Running different kinds of things

From the Powershell command line you can run different types of things:

- Another Powershell Script: type the path and name of the script file. The path is mandatory for even for the current directory (to make it difficult for malicious code to run). The file extension is optional.

```
// just use "." for the current directory e.g.
PS> .\Out-HelloWorld.ps1
hello world
Great

// The file name extension is optional.
PS> .\Out-HelloWorld
hello world
Great
```

- If a powershell script is in the Windows Path then you don't have to specfiy the full path

```
PS> Out-Pandoc.ps1 -BaseName MichaelaBanerjiAffair
```

- Another Powershell Script: In windows explorer right click the script and choose "Run with PowerShell".

  "The "Run with PowerShell" feature starts a Windows PowerShell session that has an execution policy of Bypass, runs the script, and closes the session."

  You cannot run a powershell script by double clicking it.

  (Microsoft, 2014. Scripting with Windows PowerShell) Running Scripts, https://technet.microsoft.com/en-us/library/bb613481%28v=vs.85%29.aspx

- DOS command (or "windows command-line programs"). Pass parameters using a hyphen prefix "-" or escape with double quotes. See Parameters.

```
PS> ipconfig -all

# Equivalent in this case.
PS> ipconfig -all
PS> ipconfig all    # Don't use in ISE. It chokes

# Don't use in ISE. It causes a directory intellisense popup.
# But you can use it, and sometimes must use it, by spacing past the intellisense
popup.
PS> ipconfig /all
PS> cmd /c dir /x  #
```

```
PS> ipconfig "-all"
PS> ipconfig "all"
PS> ipconfig "/all"

# Do this, to escape the parameter so it is not interpreted by Powershell
PS> Nant "-D:debug=false"
```

Parameters

- Run an exe (including those with a GUI) in Window's path variable.

```
# Opens notepad
PS> notepad
```

(Microsoft, 2014. Scripting with Windows PowerShell) Running Windows Programs, https://technet.microsoft.com/en-us/library/bb648600%28v=vs.85%29.aspx

- Run an exe (including those with a GUI) in Window's path variable. Both capture output and send results to the console.

```
$cmdOutput = openssl.exe ca -batch `
    -config $mOpenSslConfigurationFile `
    -in $mUserCertificateFileCertificateSigningRequest `
    -out $mUserCertificateFileCertificate `
    -outdir ($mUserCertificateDirectory + 'out\')  2>&1
```

(Anon., 2015) " How do I capture the output into a variable from an external process in Powershell?", answser by manojlds, 2013-04-13, http://stackoverflow.com/a/8185893/872154

- Run an exe at an arbitrary location, outside the Window's path variable.

```
# Technique 1
PS> Invoke-Item "C:\Program Files (x86)\MyLifeOrganized.net\MLO\mlo.exe";

# Technique 2
PS> &("C:\Program Files (x86)\MyLifeOrganized.net\MLO\mlo.exe");

# Technique 3
PS> &"C:\Program Files (x86)\MyLifeOrganized.net\MLO\mlo.exe";
```

Invoke-Item acts on the item as a double click would in Windows Explorer.

- Run a Java Console application

```
PS ...\TutorialAtOracle\out\production\HelloWorldViaCommandLine> Java
au.com.softmake.HelloWorldViaCommandLine Cool Running
Hello World!
Arg 0: Cool
Arg 1: Running
```

- Get-CimInstance (superceding Get-WmiObject). This accesses Windows Management Instrumentation (WMI) infrastructure.

```
PS> Get-CimInstance Win32_Bios

SMBIOSBIOSVersion : GAZ7711H.86A.0066.2013.0521.1509
Manufacturer      : Intel Corp.
Name              : GAZ7711H.86A.0066.2013.0521.1509
SerialNumber      :
Version           : INTEL - 42
```

" Starting in Windows PowerShell 3.0, this cmdlet [Get-WmiObject] has been superseded by Get-CimInstance."

(Microsoft, 2015. Core Modules in Windows PowerShell) "Get-Help Get-WmiObject", https://technet.microsoft.com/library/a3470de7-e427-4bd1-8a97-6e9d22a01da6%28v=wps.630%29.aspx "Get-Help Get-CimInstance", https://technet.microsoft.com/library/jj590758%28v=wps.630%29.aspx (Microsoft, 2015. WMI Reference) WMI Reference, https://msdn.microsoft.com/en-us/library/aa394572%28v=vs.85%29.aspx

Running considerations

There are various kinds of techniques to use when running something from powershell, to do with parameter handling and command invocation. See:

- Parameters, supplying.
- Variables in commands and expressions.
- Splatting.
- Script blocks and the Call Operator.
- Functions and the call operator.

Statements

Introduction

Powershell is a Windows shell. A shell is an interface to access an operating system's (OS) services, applications, and files.

At its heart Powershell executes statements. The statements are executed either:

- From a command-line (or "console"); or
- In a script.

(Microsoft, 2014. Scripting with Windows PowerShell) "Scripting with Windows PowerShell", https://technet.microsoft.com/en-us/library/bb978526.aspx

The statement types are:

- commands
  - cmdlets
  - functions
  - filters
  - alias
  - applications
- (powershell) scripts.
- variable assignment.

Cmdlets ("command-lets")

Cmdlets have a Verb-Noun form.

```
Get-Item
Copy-Item
Start-Service
Stop-Service
```

(Microsoft, 2014. Scripting with Windows PowerShell) Windows PowerShell Cmdlets, https://technet.microsoft.com/en-us/library/bb648597%28v=vs.85%29.aspx

They operate on objects, not text, specifically .Net framework objects!

They can hold several objects.

Parameters, supplying

Command (including cmdlet) parameters have the form "[-ParameterName] [ParameterValue]". The ParameterName can be omitted if the ParameterValue is presented in the right order according to the syntax of the command. Pass parameter values without a prefix (for Powershell native commands).

```
# "*.txt" is a parameter value
# "-Recurse" is a parameter switch, a parameter without a value
# (The value is implicitly a boolean true).
PS> Get-ChildItem -Filter *.txt -Recurse
```

When passing parameters to non Powershell commands (e.g. Dos commands):

- Ordinarily, when there is no need for escaping, use the hypen "-" prefix.

No prefix and "/" can be used but only in the native powershell command line, not the ISE. Therefore using a hypen prefix "-" best facilitates moving code between different Powershell environments.

```
# Equivalent in this case.
PS> ipconfig -all
PS> ipconfig all    # Don't use in ISE. It chokes

# Don't use in ISE. It causes a directory intellisense popup.
# But you can use it, and sometimes must use it, by spacing past the intellisense
popup.
PS> ipconfig /all
PS> cmd /c dir /x  #
```

- Invoke-Expression is handy to pass arguments that need to be calculated first.

```
$Arguments = ('x ' + ($baseName + '.docm.zip') + ' -o' + ($baseName + '\'))
Invoke-Expression "7z.exe $Arguments"
```

C:\Users\John\Documents\CustomData\Direct\Live\Microsoft\Templates\Custom Word Templates\TemplateDevelopment\jlbWordAddIn\make.ps1

- (Not recommended generally) You can use the Start-Process command, with the ArgumentList parameter, to pass arguments that need to be calculated first.

```
Start-Process -NoNewWindow 7z.exe -ArgumentList ('x ' + ($baseName + '.zip') + ' -o' +
($baseName + '\'))

Start-Process -FilePath "C:\Program Files (x86)\Microsoft
Office\root\Office16\WINWORD.EXE" -ArgumentList "/w"
```

This will hang a session unless you handle the process. So not generally recommended

C:\Users\John\Documents\CustomData\Direct\Live\Microsoft\Templates\Custom Word Templates\TemplateDevelopment\jlbWordAddIn\make.ps1

- When Powershell would start to interpret the parameter (e.g. because it finds a reserved word) you can:
    - Escape it with double quotes; or

```
# Equivalent.
PS> ipconfig "-all"
PS> ipconfig "all"
PS> ipconfig "/all"
PS> cmd "/c" "dir" "/x"

# Not this. Windows PowerShell interprets this as
# two parameters, "-D" and "debug=false"
PS> Nant -D:debug=false

# Do this, to escape the parameter so it is not interpreted by Powershell
PS> Nant "-D:debug=false"
```

(Microsoft, 2014. Scripting with Windows PowerShell) Windows Commands and Utilities, https://technet.microsoft.com/en-us/library/bb648599%28v=vs.85%29.aspx

    - Use the stop-parsing symbol (--%); or

```
# Powershell will choke
icacls X:\VMS /grant Dom\HVAdmin:(CI)(OI)F

# Use the stop-parsing symbol so that following characters will not be
# interpreted by powershell except for Windows syntax (%<Name>%) with their
values.
icacls X:\VMS --% /grant Dom\HVAdmin:(CI)(OI)F
```

```
# Powershell will send the following to the icacls program
X:\VMS /grant Dom\HVAdmin:(CI)(OI)F
```

(Microsoft, 2014. Scripting with Windows PowerShell) Running Windows Programs, https://technet.microsoft.com/en-us/library/bb648600%28v=vs.85%29.aspx

- Use backticks (`).

```
icacls X:\VMS /grant Dom\HVAdmin:`(CI`)`(OI`)F
```

(Microsoft, 2014. Scripting with Windows PowerShell) Running Windows Programs, https://technet.microsoft.com/en-us/library/bb648600%28v=vs.85%29.aspx

Outputting

Outputting options

You don't need any particular command to output. The following gets output:

- Here strings.
- Strings.
- Variables.

```
# Script
@"

###############################################################################################
OUTPUT
###############################################################################################
"@
'Hello World'
$a = 'Nice' # This line doesn't output
$a


# Output

###############################################################################################
OUTPUT
###############################################################################################
Hello World
Nice
```

(Bentley, n.d.) C:\Users\John\Documents\Sda\Code\Windows\PowerShell\Examples\Demo-Output.ps1

Commands get output.

```
PS> Get-ChildItem


    Directory: C:\Users\John\Documents\Sda\Code\Windows\PowerShell\Examples


Mode                 LastWriteTime     Length Name
----                 -------------     ------ ----
d----     2015-07-Jul-03-Fri                dir01
                           03:37
d----     2015-07-Jul-03-Fri                dir02
                           03:37
-a---     2014-05-May-04-Sun             1293 Adb-Connect.ps1 - Shortcut.lnk
                           19:19
-a---     2015-07-Jul-15-Wed             2409 Demo-Arrays.ps1
...
```

(Bentley, n.d.) 2015-07-25 15:48

Using output commands:

- Write-Output
- Write-Host.
- Out-Host.
- Add-Content

```
# Script
Write-Output 'Come at me bro';
Write-Host 'For the love of everything'
Out-Host -InputObject 'Fun'

# Output
Come at me bro
For the love of everything
Fun
```

(Bentley, n.d.) C:\Users\John\Documents\Sda\Code\Windows\PowerShell\Examples\Demo-Output.ps1

Echo substitute.

```
Set-PSDebug -Trace 1;
#.... commands
```

Suppress output via three options.

```
New-Item -ItemType Directory -Force -Path $path | Out-Null

$null = New-Item -Path c:\temp\foo -ItemType Directory

[Void](New-Item -ItemType Directory -Force -Path $path)
```

"Hide powershell output", Answer by ChiliYago, https://stackoverflow.com/a/46586504/872154
C:\Users\John\Documents\Sda\Code\Windows\PowerShell\Examples\Demo-FileSystemOperations.ps1v

Formatting

Format* commands

Formatting is achieved with only four commands:

- Format-List
- Format-Table
- Format-Custom
- Format-Wide

(Microsoft, 2014. Scripting with Windows PowerShell) Formatting Command Output, https://technet.microsoft.com/en-us/library/bb613485%28v=vs.85%29.aspx

> When you run a command, the Windows PowerShell calls the default formatter, which is determined by the type of data being displayed.

(Microsoft, 2014. Scripting with Windows PowerShell) Formatting Command Output, https://technet.microsoft.com/en-us/library/bb613485%28v=vs.85%29.aspx

You can override the default formatter by pipeing the command to a Format* command of your choice.

```
PS> Get-ChildItem | Format-Table -AutoSize

    Directory: C:\Users\John\Documents\zTemp


Mode                 LastWriteTime  Length Name
----                 -------------  ------ ----
d---- 2015-07-Jul-06-Mon     20:01         Images
d---- 2015-07-Jul-06-Mon     20:02         Stuff
-a--- 2014-11-Nov-15-Sat     15:18      11 JohnFile01
-a--- 2015-07-Jul-06-Mon     22:36      17 JohnFile01.txt
```

You can use the property parameter of the Format* command to determine which properties get displayed.

```
PS> Get-ChildItem | Format-Table -Property Name, Length -AutoSize

Name                     Length
----                     ------
Images
Stuff
JohnFile01               11
JohnFile01.txt           17

# Or Pass the properties as a positional parameter
PS> Get-ChildItem | Format-Table Name, Length -AutoSize
```

(Microsoft, 2014. Scripting with Windows PowerShell) Formatting Command Output, https://technet.microsoft.com/en-us/library/bb613485%28v=vs.85%29.aspx

Format Table Tips

Truncate (set the maximum width of) columns. You do this by providing a hash, setting each element with the keys Expression, Label, Width.

```
PS> Get-Package Windows | Sort-Object Name | Format-Table ( `
@{Expression={$_.Name};Label="Name";width=50}, `
@{Expression={$_.Version};Label="Version";width=25}, `
@{Expression={$_.ProviderName};Label="ProviderName";width=25} `
)

Name                                               Version                   ProviderName
----                                               -------                   ------------
Behaviors SDK (Windows Phone) for Visual Studio... 12.0.51210.80             msi
Behaviors SDK (Windows) for Visual Studio 2013     12.0.51210.80             msi
Blend for Visual Studio SDK for Windows Phone 8.0  3.0.30924.0               msi
Cumulative Update for Windows 10 for x64-based ...                           msu
Cumulative Update for Windows 10 for x64-based ...                           msu

PS> Get-ChildItem | Select-Object -First 4 | Format-Table (@{Expression={$_.Name};width=8})
-HideTableHeaders

.ance...
.android
.Andr...
.Andr...
```

(Bentley, n.d.) 2015-08-14 01:30
Based on: StackExchange > Superuser > " How can I shrink the width of my columns in powershell on Windows 10?" > Answer By MrStatic, http://superuser.com/a/956308

Composite Formatting

Composite formatting takes a composite format string and an object list to produce a desired string result.

```
PS> Get-ChildItem -Recurse | `
```

```
ForEach-Object {'{0,-15} ~ {1,10:N} ~ {2,20:yyyy-MM-dd hh:mm} ~ {0}' `
-f $_.Name, $_.Length, $_.LastAccessTime}

Apps            ~       1.00 ~     2015-07-13 01:41 ~ Apps
Examples        ~       1.00 ~     2015-07-24 07:36 ~ Examples
Libraries       ~       1.00 ~     2015-07-10 05:19 ~ Libraries
nisdf.txt       ~       0.00 ~     2015-07-21 08:30 ~ nisdf.txt
Connect-Adb.ps1 ~      97.00 ~     2014-05-04 07:18 ~ Connect-Adb.ps1
Get-PathSlashFileName.ps1 ~     473.00 ~     2013-01-28 07:30 ~ Get-PathSlashFileName.ps1
```

(Microsoft, 2015..NET Framework Class Library) Composite Formatting, https://msdn.microsoft.com/en-us/library/txafckwd%28v=vs.110%29.aspx
(Bentley, n.d.) 2015-07-26 18:08

Composite formatting syntax.

```
{index[,alignment][:formatString]}

index.
      A zero-based index of the object list. An object can be referenced multiple times.

alignment.
      An integer indicating the preferred field width. Spaces will be pad the result if the
      field is less than the alignment specificed.
            Postive integer  = right aligned.
            Negative integer = left aligned.

formatString.
      The .net standard or custom format string.
```

(Microsoft, 2015..NET Framework Class Library) Composite Formatting, https://msdn.microsoft.com/en-us/library/txafckwd%28v=vs.110%29.aspx

Composite formatting works with single quotes (') or double quotes ("), and variable parsing is respected according to the normal rules.

```
# Single quote: Perform composite substitution, don't interpret variables.
PS> '$LASTEXITCODE {0} indicates an error' -f $LASTEXITCODE
$LASTEXITCODE 16 indicates an error

# Double quote: Perform composite substitution, interpret variables.
PS> "$LASTEXITCODE {0} indicates an error" -f $LASTEXITCODE
16 16 indicates an error
```

(Bentley, n.d.) 2015-08-03 22:57

To obtain the .net standard or custom format string codes see (Bentley, 2015. dotNet Format Strings, \\ATLAS\Documents\Sda\Info\DotNet\KB\Quick Reference\Code\Operators And Special Characters\dotNet Format Strings.docx)

Composite String formatting can be done from at least the following:

- Powershell –f operator.
- Dotnet. String::Format()
- Dotnet. Console::Writeline().
- Dotnet. StringBulder.AppendFormat().

```
# Powershell -f operator.
PS> Get-ChildItem | foreach { "{0,25:f5} xx" -f $_.Length }

# Dotnet. String::Format()
PS> Get-ChildItem | foreach { [String]::Format("{0,25:f5} xx", $_.Length) }

# Dotnet. Console::Writeline().
PS> Get-ChildItem | foreach { [Console]::WriteLine("{0,25:f5} xx", $_.Length) }
```

```
# Dotnet. StringBulder.AppendFormat().
$sb = New-Object -TypeName System.Text.StringBuilder
Get-ChildItem | foreach { $sb.AppendFormat("{0}`r`n", $_.Name) } | Out-Null
$sb.ToString()
```

(Bentley, n.d.) 2015-07-26 22:01

Object Formatting (non composite)

To return the value of the object instance's property, just directly refer to the property name. ...

```
# Get object property value
PS> Get-ChildItem | ForEach-Object BaseName
countries
debug
file01
file02
file03
file04
LICENSE
```

(Bentley, n.d.) 2017-02-24 18:00

To return the value of the object instance's method, use:
* ForEach-Object;
* the item syntax ($_.Property or $_.Method()); and
* Surrounded by brackets "{}" to evaluate as an expression.

```
# Get Method Return Values
PS> Get-Content countries.txt | ForEach-Object {$_.ToString()}

Afghanistan
Albania
Algeria
Andorra
```

(Bentley, n.d.) 2017-02-24 18:00

The following truncates string output.

```
ps> Get-Content countries.txt | ForEach-Object {$_.Substring(0,4)}
Afgh
Alba
Alge
Ando

ps> Get-Content countries.txt | ForEach-Object {$_[0..3] -join ""}
Afgh
Alba
Alge
And

ps> Get-ChildItem | ForEach-Object {$_.Name.Substring(0,4)}
.git
coun
debu
file

ps> Get-ChildItem | Format-Table @{Expression={$_.Name};Label="FileName";width=4}
File
Name
----
....
c...
d...
f...
```

(Bentley, n.d.) 2017-02-19 16:12

Only the value of one of many properties of an object.

A command may return an object with many properties. You can return one of the properties, either ...

... the key/value pair of the property ...

```
PS> Invoke-WebRequest -Uri http://localhost:8080/

StatusCode        : 200
StatusDescription : OK
Content           : <!DOCTYPE html PUBLIC "-//W3C//DTD XHTML 1.1//EN"
                      "http://www.w3.org/TR/xhtml11/DTD/xhtml11.dtd">
                    <!--
                      Omit XML declaration from top line as it puts IE6 in quirks mode.
                      Since the XML declar...
RawContent        : HTTP/1.1 200 OK
[etc..]

PS> Invoke-WebRequest -Uri http://localhost:8080/ | Format-List -Property Content

Content : <!DOCTYPE html PUBLIC "-//W3C//DTD XHTML 1.1//EN"
            "http://www.w3.org/TR/xhtml11/DTD/xhtml11.dtd"> [etc..]
```

... or just the value of one of these properties ..

```
PS> Invoke-WebRequest -Uri http://localhost:8080/ | Select-Object -ExpandProperty Content

<!DOCTYPE html PUBLIC "-//W3C//DTD XHTML 1.1//EN"
  "http://www.w3.org/TR/xhtml11/DTD/xhtml11.dtd"> [etc..]
```

(Bentley, n.d.) 2016-05-18 15:22
See http://superuser.com/a/994132/226936

Paging

When you more than one page of output to display you can halt output one page at a time. This is called "paging". Use `Out-Host -Paging` or `more`.

```
# Recommended, it provides the following paging instructions:
# <SPACE> next page; <CR> next line; Q quit
PS> Get-Command | Out-Host -Paging

# Not recommended, it doesn't provide paging instructions
PS> Get-Command | more
```

Paging does not work in Powershell ISE. Neither for "Out-Host -Paging" nor "more".

(Microsoft, 2014. Scripting with Windows PowerShell) Redirecting Data with Out- Cmdlets,*
https://technet.microsoft.com/en-us/library/dd347585.aspx
(Bentley, n.d.) 2015-07-25 16:18

Redirection

Overview

You can redirect output, from the default command window, to either:

- A file; or
- The "success output stream".

(Microsoft, 2015. Core Modules in Windows PowerShell) about_Redirection, https://technet.microsoft.com/en-us/library/hh847746.aspx

Three redirection techniques:

1. Out-File cmdlet. Use this if you need to avail yourself of the parameters of that command. e.g. Encoding, Force, Width, or NoClobber.
2. Tee-Object. Sends output to a text file AND sends it to the pipeline.
3. Add-Content.
4. Redirection Operators.

(Microsoft, 2015. Core Modules in Windows PowerShell) about_Redirection, https://technet.microsoft.com/en-us/library/hh847746.aspx

Redirection Operators

Redirection operator syntax:

```
<input> <operator> [<path>\]<file>
```

(Microsoft, 2015. Core Modules in Windows PowerShell) about_Redirection, https://technet.microsoft.com/en-us/library/hh847746.aspx

Basic Examples

```
# Redirect to a file in the current directory
PS> Get-ChildItem > dump.txt

# Append to a file in the current directory
PS> Get-ChildItem >> dump.txt
```

Redirection operator levels:

Level	Meaning
[omitted]	Standard level
&1	Success output
2	Errors
3	Warning messages
4	Verbose output
5	Debug messages
*	All output

(Microsoft, 2015. Core Modules in Windows PowerShell) about_Redirection, https://technet.microsoft.com/en-us/library/hh847746.aspx

All redirection operator meanings.

Operator	Description	Example
>	Sends output to the specified file.	Get-Process > Process.txt

>>	Appends the output to the contents of the specified file.	dir *.ps1 >> Scripts.txt
2>	Sends errors to the specified file.	Get-Process none 2> Errors.txt
2>>	Appends errors to the contents of the specified file.	Get-Process none 2>> Save-Errors.txt
2>&1	Sends errors (2) and success output (1) to the success output stream.	Get-Process none Powershell 2>&1
3>	Sends warnings to the specified file.	Write-Warning "Test!" 3> Warnings.txt
3>>	Appends warnings to the contents of the specified file.	Write-Warning "Test!" 3>> Save-Warnings.txt
3>&1	Sends warnings (3) and success output (1) to the success output stream.	function Test-Warning { Get-Process PowerShell;Write-Warning "Test!" }; Test-Warning 3>&1
4>	Sends verbose output to the specified file.	Import-Module * -Verbose 4> Verbose.txt
4>>	Appends verbose output to the contents of the specified file.	Import-Module * -Verbose 4>> Save-Verbose.txt
4>&1	Sends verbose output (4) and success output (1) to the success stream.	output Import-Module * -Verbose 4>&1
5>	Sends debug messages to the specified file.	Write-Debug "Starting" 5> Debug.txt
5>>	Appends debug messages to the contents of the specified file.	Write-Debug "Saving" 5>> Save-Debug.txt
5>&1	Sends debug messages (5) and success output (1) to the success output stream.	function Test-Debug { Get-Process PowerShell Write-Debug "PS" }; Test-Debug 5>&1
*>	Sends all output types to the specified file.	{ Get-Process PowerShell, none Write-Warning "Test!" Write-Verbose "Test Verbose" Write-Debug "Test Debug" } Test-Output *> Test-Output.txt Test-Output *>> Test-Output.txt Test-Output *>&1
*>>	Appends all output types to the contents of the specified file.	[As Above]
*>&1	Sends all output types (*) to the success output stream.	[As Above]

(Microsoft, 2015. Core Modules in Windows PowerShell) about_Redirection, https://technet.microsoft.com/en-us/library/hh847746.aspx

Inputting

Use Read-Host to get user input.

```
<#
.SYNOPSIS
    Prompts the user for input and assigns that input to the variable passed by reference.

.DESCRIPTION
    Prompts the user for input and assigns that input to the variable passed by reference.
    If the user doesn't input anything (by hitting enter) then the supplied default value is
assigned
    to the variable.
```

```
    If the user enters a period ".", then $null is assigned to the variable.

.PARAMETER VariableToSet
    The variable that you want to assign input results to. This must be passed by reference.
    This must be declared before beign passed by reference.

.PARAMETER PromptString
    The message to the user to prompt them for input.

.PARAMETER VariableDefault
    The value which is assigned to the variable passed by reference, if the user doesn't
input
    a value (when they hit Enter).

.EXAMPLE
    $CountryName = $null;Set-VariableFromInput `
        -VariableToSet ([ref]$CountryName) `
        -PromptString "Country Name (2 letter code). E.g. [US] Default = " `
        -VariableDefault "AU"

    Assigns user input to $CountryName.
    If the user hits Enter then $CountryName will equal "AU".
    If the user hits "."   then $CountryName will equal $null.
    Note $CountryName has been declared before being passed by reference to Set-
VariableFromInput.
#>
function Set-VariableFromInput {
    param (
      [parameter(Mandatory=$true)]
      [ref]
      $VariableToSet,
      [parameter(Mandatory=$true)]
      [String]
      $PromptString,
      [parameter(Mandatory=$true)]
      [String]
      $VariableDefault
    )

    $VariableToSet.Value = Read-Host -Prompt ($PromptString + "[$VariableDefault]")

    If ($VariableToSet.Value -eq ".") {
        $VariableToSet.Value = $null
    } elseif ($VariableToSet.Value -eq "") {
        $VariableToSet.Value= $VariableDefault
    }

}

$CountryName = $null
Set-VariableFromInput `
    -VariableToSet ([ref]$CountryName) `
    -PromptString "Country Name (2 letter code). E.g. [US] Default = " `
    -VariableDefault "AU"
"Result $CountryName"

# Run the above script
PS> . .\Demo-Input.ps1
Country Name (2 letter code). E.g. [US] Default = [AU]: sdf
Result sdf
```

(Bentley, n.d.) 2015-09-16 18:18 \\ATLAS\Documents\Sda\Code\Windows\PowerShell\Examples\Demo-Input.ps1

Get Information (Discovery and Help)

Usage

To get information on a single command you generally supply a single command as a parameter to the relevant informational command.

```
# Explicitly named parameter
Get-Help -Name Get-ChildItem
```

```
Get-Command -Name Get-ChildItem
Get-Alias -Name gci
Get-Alias -Definition Get-ChildItem  # Particular parameter is mandatory (despite docs).

# Implicit parameter
Get-Help Get-ChildItem
Get-Command Get-ChildItem
Get-Alias gci
```

To discover a range of commands, supply a wildcard, or supply nothing to return everything.

```
Get-Help Get*
Get-Help about*
Get-Command *Process
Get-Command
Get-Alias
Get-Variable

# Only useful to return all
Get-PSDrive
Get-PSProvider
```

Get-Help can be accessed with a shortcut "-?" supplied as a parameter to any command

```
Get-ChildItem -?
```

"help" is a function that calls Get-Help, effectively serving as an alias. "man" is an alias for "help".

```
# Equivalent.
Get-Help Get-ChildItem
help Get-ChildItem
man Get-ChildItem
```

(Microsoft, 2014. Scripting with Windows PowerShell) Getting Detailed Help Information, https://technet.microsoft.com/en-us/library/dd347689.aspx

Launch online help from the console.

```
get-help get-process -online
```

(Microsoft, 2014. Scripting with Windows PowerShell), Getting Help: Get-Help, https://technet.microsoft.com/en-us/library/bb648604%28v=vs.85%29.aspx

GetStatement discovery and help

- Via the following specific Powershell commands.
  - Get-Help
  - Get-Command
  - Get-Alias
  - Get-Member
  - Get-Variable
  - Get-PSDrive
  - Get-PSProvider

(Microsoft, 2015. Core Modules in Windows PowerShell) Using Functions, https://technet.microsoft.com/en-us/library/dd745030%28v=vs.85%29.aspx

- Via Powershell ISE > View > Show Commands Add-in.

- Online via Microsoft Technet > Core Modules in Windows PowerShell.
  - > Windows PowerShell 5.0, https://technet.microsoft.com/en-us/library/hh847741.aspx
  - > Windows PowerShell 5.0 > Microsoft.PowerShell.Core Module, https://technet.microsoft.com/en-us/library/hh847840.aspx

(Microsoft, 2015. Core Modules in Windows PowerShell) Core Modules in Windows PowerShell, https://technet.microsoft.com/en-us/library/mt156967.aspx

Concept discovery and help

- Via the Powershell command.
  - Get-Help about*
- Online via Microsoft Technet > Core Modules in Windows PowerShell.
  - > Windows PowerShell 5.0 > Microsoft.PowerShell.Core Module > Core About Topics, https://technet.microsoft.com/en-us/library/hh847856.aspx

(Microsoft, 2015. Core Modules in Windows PowerShell) Core Modules in Windows PowerShell, https://technet.microsoft.com/en-us/library/mt156967.aspx

Function discovery and help

```
# Discovery
Get-Command -CommandType Function # All available functions.

&{ Set-Location function: ; Get-ChildItem } # find all functions in your session


# Help
Get-Help Out-Speech
```

(Microsoft, 2015. Core Modules in Windows PowerShell) Using Functions, https://technet.microsoft.com/en-us/library/dd745030%28v=vs.85%29.aspx

Object discovery and help

For Get-Member pipe the Get* command in.

```
# Get all members of the piped in command
Get-ChildItem | Get-Member

# Get only those members that match the wild
Get-ChildItem | Get-Member -Name *time*
```

To get member information for a .Net Object instance.

```
$sb = New-Object -TypeName System.Text.StringBuilder
PS> $sb | Get-Member


   TypeName: System.Text.StringBuilder

Name           MemberType            Definition
----           ----------            ----------
Append         Method                System.Text.StringBuilder Append(char v...
AppendFormat   Method                System.Text.StringBuilder AppendFormat(...
AppendLine     Method                System.Text.StringBuilder AppendLine(),...
Clear          Method                System.Text.StringBuilder Clear()
```

To discover the static members of a class use the -Static parameter of Get-Member.

```
PS> [System.Environment] | Get-Member -Static
```

```
   TypeName: System.Environment

Name                        MemberType Definition
----                        ---------- ----------
Equals                      Method     static bool Equals(System.Object objA,...
Exit                        Method     static void Exit(int exitCode)
ExpandEnvironmentVariables Method      static string ExpandEnvironmentVariabl...
FailFast                    Method     static void FailFast(string message), ...
...
CommandLine                 Property   static string CommandLine {get;}
CurrentDirectory            Property   static string CurrentDirectory {get;set;}
CurrentManagedThreadId      Property   static int CurrentManagedThreadId {get;}
ExitCode                    Property   static int ExitCode {get;set;}
...
```

(Microsoft, 2014. Scripting with Windows PowerShell) Using Static Classes and Methods, https://technet.microsoft.com/en-us/library/dd347632.aspx

To get all property values for a particular object instance:

- Pipe the object into a format-list and filter by all properties; or

```
PS> Get-ChildItem -File JohnFile01.txt | Format-List -Property *

...
BaseName          : JohnFile01
Mode              : -a---
Name              : JohnFile01.txt
Length            : 17
DirectoryName     : C:\Users\John\Documents\zTemp
Directory         : C:\Users\John\Documents\zTemp
IsReadOnly        : Falsev
...

PS> Get-Service Spooler | Format-List -Property *

Name                : Spooler
RequiredServices    : {RPCSS, http}
CanPauseAndContinue : False
CanShutdown         : False
CanStop             : True
DisplayName         : Print Spooler
DependentServices   : {Fax}
MachineName         : .
ServiceName         : Spooler
ServicesDependedOn  : {RPCSS, http}
...
```

- Get a specific value for a specific property.

```
PS> (Get-ChildItem -File JohnFile01.txt).Length
17

PS> (Get-Service spooler).CanStop
True
```

(Microsoft, 2014. Scripting with Windows PowerShell) Learning About Objects: Get-Member, https://technet.microsoft.com/en-us/library/bb613480%28v=vs.85%29.aspx

You can also call methods using the `(get-service <service-name>).<method-name>()`.

```
(get-service schedule).stop()
```

(Microsoft, 2014. Scripting with Windows PowerShell) Learning About Objects: Get-Member, https://technet.microsoft.com/en-us/library/bb613480%28v=vs.85%29.aspx

For arrays, to get the members for each object type in the array use piping.

```
PS> $a = 1, 2, "Cool"
PS> $a | Get-Member

   TypeName: System.Int32

Name        MemberType Definition
----        ---------- ----------
CompareTo   Method     int CompareTo(System.Object value), int CompareTo(int value)...
Equals      Method     bool Equals(System.Object obj), bool Equals(int obj), bool
IEquatable[int].Equals(int other)
GetHashCode Method     int GetHashCode()
GetType     Method     type GetType()
GetTypeCode Method     System.TypeCode GetTypeCode(), System.Type
...

   TypeName: System.String

Name             MemberType            Definition
----             ----------            ----------
Clone            Method                System.Object Clone(), System.Object ...
CompareTo        Method                int CompareTo(System.Object value), ...
Contains         Method                bool Contains(string value)

...
```

(Microsoft, 2014. Scripting with Windows PowerShell) Creating .NET and COM Objects (New-Object), https://technet.microsoft.com/en-us/library/dd347574.aspx

For arrays, to get the members of the array object itself use the -InputObject parameter of Get-Member.

```
PS> $a = 1, 2, "Cool"
PS> Get-Member -InputObject $a

   TypeName: System.Object[]

Name           MemberType            Definition
----           ----------            ----------
Count          AliasProperty         Count = Length
Add            Method                int IList.Add(System.Object value)
Address        Method                System.Object&, mscorlib, Version=4.0.0.0,...
Clear          Method                void IList.Clear()
Clone          Method                System.Object Clone(), System.Object ICloneable.Clone()
...
```

(Microsoft, 2014. Scripting with Windows PowerShell) Creating .NET and COM Objects (New-Object), https://technet.microsoft.com/en-us/library/dd347574.aspx

Parameter discovery and help

A property that an object returns from a command is distinct from the parameter you supply to a command.

Get-Help on a particular parameter.

```
PS> Get-Help Get-ChildItem -Parameter Recurse
```

Get-Help on all parameters

```
PS> Get-Help Get-ChildItem -Parameter *
```

Parameters as a list from Get-Command.

```
PS> (Get-Command Get-ChildItem).Parameters | Format-Table -Property Key
```

```
Key
---
Path
LiteralPath
Filter
Include
Exclude
Recurse
Force
Name
....
```

(Microsoft, 2014. Scripting with Windows PowerShell) Using Cmdlet Parameters, https://technet.microsoft.com/en-us/library/dd745032%28v=vs.85%29.aspx

Data Types

Examine Type

Examine type.

```
# GetType() Technique
PS> $x = 12

PS> $x.GetType().FullName
System.Int32

# Pipe to Get-Member Technique
PS> $x | Get-Member


   TypeName: System.Int32

Name        MemberType Definition
----        ---------- ----------
CompareTo   Method     int CompareTo(System.Object value), int CompareTo(int...
Equals      Method     bool Equals(System.Object obj), bool Equals(int obj),...
GetHashCode Method     int GetHashCode()
GetType     Method     type GetType()
GetTypeCode Method     System.TypeCode GetTypeCode(), System.TypeCode IConve...
```

(Bentley, n.d.) 2015-07-30 09:36

Base Data Types

Base Data types. DotNet mapping.

Category	Class name	Description	PowerShell	Visual Basic data type	C# data type
Integer	Byte	An 8-bit unsigned integer.	byte	Byte	byte
	SByte	An 8-bit signed integer. Not CLS-compliant.		SByte	sbyte
	Int16	A 16-bit signed integer.		Short	short
	Int32	A 32-bit signed integer.	int	Integer	int
	Int64	A 64-bit signed integer.	long	Long	long
	UInt16	A 16-bit unsigned integer. Not CLS-compliant.		UShort	ushort

	UInt32	A 32-bit unsigned integer. Not CLS-compliant.		UInteger	uint
	UInt64	A 64-bit unsigned integer. Not CLS-compliant.		ULong	ulong
Floating point	Single	A single-precision (32-bit) floating-point number.	single	Single	float
	Double	A double-precision (64-bit) floating-point number.	double	Double	double
Logical	Boolean	A Boolean value (true or false).		Boolean	bool
Other	Char	A Unicode (16-bit) character.	char	Char	char
	Decimal	A decimal (128-bit) value.	decimal	Decimal	decimal
	IntPtr	A signed integer whose size depends on the underlying platform (a 32-bit value on a 32-bit platform and a 64-bit value on a 64-bit platform).		IntPtr No built-in type.	IntPtr No built-in type.
	UIntPtr	An unsigned integer whose size depends on the underlying platform (a 32- bit value on a 32-bit platform and a 64-bit value on a 64-bit platform). Not CLS-compliant.		UIntPtr No built-in type.	UIntPtr No built-in type.
Class objects	Object	The root of the object hierarchy.		Object	object
	String	An immutable, fixed-length string of Unicode characters.	string	String	string

Modifiled From:
(Microsoft, 2015..NET Framework Class Library), Development Guide, .NET Framework Class Library Overview, System Namespace, https://msdn.microsoft.com/en-us/library/hfa3fa08%28v=vs.110%29.aspx
(Sheppard, 2015. Windows PowerShell command line syntax) PowerShell Data Types, http://ss64.com/ps/syntax-datatypes.html

Other common datatypes

Other common Powershell/.Net Data types.

.Net	Powershell
[System.Array]	array
[System.Collections.Hastable]	hashtable
[System.DateTime]	datetime
[System.Xml.XmlDocument]	xml
[System.Text.RegularExpression.Regex]	regex

(Sheppard, 2015. Windows PowerShell command line syntax) PowerShell Data Types, http://ss64.com/ps/syntax-datatypes.html

Loose typing

By default variables are loosely typed, they accept the type of whatever value is assigned to them.

```
$a = 12     (System.Int32)
$a = "Word" (System.String)
$a = 12, "Word" (System.Int32, System.String)
$a = dir C:\Windows\System32  (Files and folders)
```

(Microsoft, 2015. Core Modules in Windows PowerShell) about_Variables, https://technet.microsoft.com/en-us/library/hh847734.aspx

Strong typing

You can make the variable strongly typed by prefixing it, on assignment, with a type in brackets. That will either cast the value to the type if possible or raise an error.

```
[int]$a = 45 # Varible now has Int32 datatype.
$a = "1234" # Cast String to an Int32
$a = "nice" # Error

[datetime]$birthdate = "2000-06-01" # Cast String to a datetime

$birthdate
# Output
Thursday, 1 June 2000 00:00:00
```

(Microsoft, 2015. Core Modules in Windows PowerShell) about_Variables, https://technet.microsoft.com/en-us/library/hh847734.aspx
C:\Users\John\Documents\Sda\Code\Windows\PowerShell\Examples\Demo-Variables.ps1

Type specification/Type Casting

To specify a particular type you can prefix the type name in brackets ([]) or use a literal type suffix. Using a literal type suffix is not recommended as it has limited application. If you don't supply a type specifier Powershell chooses a default: [int] for an integer literal; [double] for a floating point literal; [string] for any string literal, including a single literal character.

.Net Data Type	Data type	Literal type character suffix	Literal type character example	Powershell type prefix	Powershell Type Specifier Example
System.Byte	byte			[byte]	$p = [byte]33
System.Int32	int	(None)	$x = 12	[int], [int32]	$q = [int]45
System.Int64	long	L	$y = 34L	[long], [int64]	$s = [long]44
System.Decimal	decimal	D	$z = 160D	[decimal]	$t = [decimal]150
System.Single	single			[single]	$u = [single]45.12
System.Double	double	(None)	$a = 345.78	[double]	$t = [double]45.67
System.Char	char			[char]	$v = [char]'g'
System.String	string	(None)	$b = 'r'	[string]	$b = [string]'r'

(Bentley, n.d.) 2015-07-30 10:27

Powershell type prefixes work either with our without a space after them.

```
# Without space
PS> $a = [int]44
PS> $a.GetType().FullName
System.Int32
```

```
# With Space.
PS> $d = [long] 66
PS> $d.GetType().FullName
System.Int64
```

(Bentley, n.d.) 2015-07-31 11:12

You can type cast from one variable to another.

```
# Create a variable with a type.
PS> $a = [int]44
PS> $a.GetType().FullName
System.Int32

# By default assigning one variable to another results in the new variable inheriting
# the type.
PS> $b = $a
PS> $b.GetType().FullName
System.Int32
Accessed 2015-07-31
# Cast the variable.
PS> $c = [long] $a
PS> $c.GetType().FullName
System.Int64
```

(Bentley, n.d.) 2015-07-31 11:15

Strings

Concatenation

Concatenate strings with:

- The plus (+) or plus equals (+=) operator; or
- The `-Join` operator (which joins the elements of an array).

```
# Plus (+) operator
PS> 'Now' + ' then'
Now then

# join operator
PS> $a = "Far", 'and', 'away'
PS> $a -join " "
Far and away Here Strings
```

(Sheppard, 2015. Windows PowerShell command line syntax) Escape characters, Delimiters and Quotes, http://ss64.com/ps/syntax-esc.html
(Bentley, n.d.) 2015-07-29 17:41

Here Strings

Here strings are created by booking lines of strings with an at symbol (@) and quote, double or single. Double quotes interpret variables.

```
$x = 12

@'
My here string inside single quotes
doesn't interpret variables $x
'@

# Output
My here string inside single quotes
```

```
doesn't interpret variables $x


@"
My here string inside single quotes
does interpret variables $x
"@

# Output
My here string inside single quotes
does interpret variables 12
```

(Bentley, n.d.) 2015-07-29 10:15
(Sheppard, 2015. Windows PowerShell command line syntax) Escape characters, Delimiters and Quotes, http://ss64.com/ps/syntax-esc.html

You can assign Here Strings to variables.

```
# Play.ps1
function Main() {
    $x = @'
Some here string
For you
'@
    $x
}

Main;

#Output
PS> .\Play01.ps1
Some here string
For you
```

(Bentley, n.d.) \\ATLAS\Documents\Sda\Code\Windows\PowerShell\Examples\Play01.ps1

Special Characters

Special Characters in Strings uses the backtick (`)...

Characters	Meaning
`0	Null
`a	Alert
`b	Backspace
`f	Form feed
`n	New line
`r	Carriage return
'r'n	Carriage return + New line
`t	Horizontal tab
`v	Vertical tab
--%	Stop parsing

(Microsoft, 2015. Core Modules in Windows PowerShell) about_Special_Characters, https://technet.microsoft.com/en-us/library/hh847835.aspx

Special Characters example use.

```
PS> "Now `r`n this"
Now
 this
```

(Bentley, n.d.) 2015-07-31 11:22

Select-String

Select-String operates on input strings, objects, or files and returns (by default) only those lines matching the Regex pattern.

Select-String on input strings.

```
# Usual (send it through to a non powershell dos command).
PS> ipconfig -all

Windows IP Configuration

   Host Name . . . . . . . . . . . . : Atlas
   Primary Dns Suffix  . . . . . . . :
   Node Type . . . . . . . . . . . . : Hybrid
   IP Routing Enabled. . . . . . . . : No
   WINS Proxy Enabled. . . . . . . . : No

Ethernet adapter Bluetooth Network Connection:

   Media State . . . . . . . . . . . : Media disconnected
   Connection-specific DNS Suffix  . :
   Description . . . . . . . . . . . : Bluetooth Device (Personal Area Network) #2
   Physical Address. . . . . . . . . : 00-02-72-32-7F-18
   DHCP Enabled. . . . . . . . . . . : Yes
   Autoconfiguration Enabled . . . . : Yes

# Using Select-String
# "-Pattern" is optional
PS> ipconfig -all | Select-String -Pattern DNS

   Primary Dns Suffix  . . . . . . . :
   Connection-specific DNS Suffix  . :
   ...

# Use a regex expression by escaping it.
PS> ipconfig -all | Select-String -Pattern "WINS|DHCP"

   WINS Proxy Enabled. . . . . . . . : No
   DHCP Enabled. . . . . . . . . . . : Yes
```

(Microsoft, 2014. Scripting with Windows PowerShell) Windows Commands and Utilities, https://technet.microsoft.com/en-us/library/bb648599%28v=vs.85%29.aspx
(Microsoft, 2015. Core Modules in Windows PowerShell) Select-String, https://technet.microsoft.com/library/43d70212-90e0-4a89-a148-04d7db9a7ab1%28v=wps.630%29.aspx

Select-String on files.

```
PS> Get-ChildItem C:\Users\John\Documents\zTemp -Recurse | Select-String fluff

C:\Users\John\Documents\zTemp\JohnFile01.txt:2:Fluff
C:\Users\John\Documents\zTemp\JohnFile02:1:Fluff and bump
C:\Users\John\Documents\zTemp\JohnFile02.txt:1:Fluff and bump
C:\Users\John\Documents\zTemp\JohnFile03:2:Fluff
```

(Microsoft, 2015. Core Modules in Windows PowerShell) Select-String, https://technet.microsoft.com/library/43d70212-90e0-4a89-a148-04d7db9a7ab1%28v=wps.630%29.aspx

Pattern Matching

Wildcards

Wildcard syntax.

Wildcard	Description	Example	Match	No match
*	Matches zero or more characters	A*	A, ag, apple	Banana
?	Matches exactly one character in the specified position	?N	An, in, on	Ran
[ ]	Matches a range of characters	[a-l]ook	Book, cook, look	Took
[ ]	Matches specified characters	[bc]ook	Book, cook	Hook

(Microsoft, 2015. Core Modules in Windows PowerShell) about_Wildcards, https://technet.microsoft.com/en-us/library/hh847812.aspx

Wildcard characters can be used:

- By many cmdlet parameters. Check parameter help value "Accept Wildcard Characters".
- In string comparisons in script blocks.

```
# Use in a cmdlet parameter
PS> Get-ChildItem -Path Demo* -Name

Demo-Arrays.ps1
Demo-CommentBasedHelp.ps1
Demo-ControlFlow.ps1
Demo-Functions.ps1
...

# Use in a script block string comparison
PS> Get-ChildItem -Recurse | `
Where-Object {$_.DirectoryName -notlike "*\Libraries*"} | `
Format-Table Name, DirectoryName -AutoSize

Apps
Examples
Libraries
nisdf.txt                 C:\Users\John\Documents\Sda\Code\Windows\PowerShell
Connect-Adb.ps1           C:\Users\John\Documents\Sda\Code\Windows\PowerShell\Apps
Get-PathSlashFileName.ps1 C:\Users\John\Documents\Sda\Code\Windows\PowerShell\Apps
```

(Microsoft, 2015. Core Modules in Windows PowerShell) about_Wildcards, https://technet.microsoft.com/en-us/library/hh847812.aspx
(Bentley, n.d.) 2015-07-28 16:09

Regex

Powershell regular expressions ("regexes") are .net regular expressions. See (Bentley, 2015. dotNet Framework Regular Expression Quick Reference, \\ATLAS\Documents\Sda\Info\DotNet\KB\Quick Reference\Code\Operators And Special Characters\dotNet Framework Regular Expression Quick Reference.docx)

The Powershell regular expression operators are:

```
-match, -notmatch, -replace
```

See String Operators

Powershell regex matching.

```
# Basic regex matching
```

```
PS> 'Lorem 1234 ipsum' -match '\s\d{4}\s'
True

PS> 'Lorem 1234 ipsum' -match '\s\d{3}\s'
False

# Use regex matching in a Where-Object block
PS> Get-ChildItem -Recurse -File | Where-Object { $_.Name -match '^\w{4}-' } | Format-Table
Name

Name
----
Demo-Arrays.ps1
Demo-CommentBasedHelp.ps1
Demo-ControlFlow.ps1
...
Hide-PowerShell.ps1
```

(Bentley, n.d.) 2015-07-29 08:52
(Sheppard, 2015. Windows PowerShell command line syntax) http://ss64.com/ps/syntax-regex.html
(Microsoft, 2015. Core Modules in Windows PowerShell) about_Regular_Expressions, https://technet.microsoft.com/en-us/library/hh847880.aspx

Powershell regex replacing.

```
# Basic regex replacement
PS> 'Lorem 1234 ipsum' -replace '\s\d{4}\s', ' XXXX '
Lorem XXXX ipsum

#Use capture groups.
PS> 'Now    This' -replace '(\w+)(\s+)(\w+)', '$1,$3'
Now,This

# Use regex replacement in Format-Table block
PS> Get-ChildItem -Recurse -File | Where-Object { $_.Name -match '^\w{4}-' } `
| Format-Table { $_.Name -replace '^\w{4}-', 'Cool-'}

 $_.Name -replace '^\w{4}-', 'Cool-'
-----------------------------------
Cool-Arrays.ps1
Cool-CommentBasedHelp.ps1
Cool-ControlFlow.ps1
...

# The same result is better achieved by putting the repeated regex pattern in a variable
PS> $regexPattern = '^\w{4}-'

PS> Get-ChildItem -Recurse -File | Where-Object { $_.Name -match $regexPattern } `
| Format-Table { $_.Name -replace $regexPattern, 'Cool-'}
```

(Bentley, n.d.) 2015-07-29 09:11

Partial Value Matching

Some parameters accept neither wildcards nor regexes. However, they might accept a partial value match.

```
# Wilcard not accepted
PS> Get-Package -Name Xenu*
Get-Package : No package found for  'Xenu*'.

# Partial value accepted
PS> Get-Package -Name Xenu

Name                           Version          Source           Summary
----                           -------          ------           -------
Xenu's Link Sleuth             1.3.8
```

(Bentley, n.d.) 2015-08-11 15:05

Expressions

Arithmetic expressions are possible.

```
PS> 20 + 10
30
```

(Microsoft, 2014. Scripting with Windows PowerShell) Running Commands, https://technet.microsoft.com/en-us/library/bb613482%28v=vs.85%29.aspx

Powershell understands computer units.

```
PS> 7GB / 1028KB
7140.10894941634
```

(Microsoft, 2014. Scripting with Windows PowerShell) Running Commands, https://technet.microsoft.com/en-us/library/bb613482%28v=vs.85%29.aspx

Variables

Basics

Types of variables:

- User-Created.
- Automatic: Powershell environmental variables, not changeable by the user.
- Preference: Powersshell environmental variables, changeable by the users.

Return a list of all variables, alive to the powershell session.

```
get-variable

# Or
Set-Location variable:
Get-ChildItem
```

(Microsoft, 2015. Core Modules in Windows PowerShell) about_Variables, https://technet.microsoft.com/en-us/library/hh847734.aspx

User Created Variables

You don't declare variables. Just set them with a "$" prefix.

```
$MyVariable = 1, 2, 3
$path = "C:\Windows\System32"
$processes = Get-Process
$Today = (Get-Date).date
```

(Microsoft, 2015. Core Modules in Windows PowerShell) about_Variables, https://technet.microsoft.com/en-us/library/hh847734.aspx

Display just by typing.

```
$MyVariable

#Result
1
2
3
```

Set a new value.

```
$MyVariable = "Cool"

# New result
Cool
```

Clear the value of a variable, but keep the variable itself.

```
$MyVariable = $null

# Or
Clear-Variable -name MyVariable
```

(Microsoft, 2015. Core Modules in Windows PowerShell) about_Variables, https://technet.microsoft.com/en-us/library/hh847734.aspx

Delete the variable itself.

```
remove-variable -name MyVariable

# Or
remove-item -path variable:\myvariable
```

(Microsoft, 2015. Core Modules in Windows PowerShell) about_Variables, https://technet.microsoft.com/en-us/library/hh847734.aspx

Variables in commands and expressions

To use a variable in a command or expression: use with the $ prefix; and/or do so in double quotation marks "", as that will interpret the variable. If you put the variable in single quotaton marks '' it will not be interpreted as a variable.

```
# Open the Powershell profile (the file) in notepad for editing
notepad $profile
notepad "$profile"  # Interpreted as variable.

# Open notepad with window named '$profile'
notepad '$profile' # Interpreted as a literal, not a variable.
```

(Microsoft, 2015. Core Modules in Windows PowerShell) about_Variables, https://technet.microsoft.com/en-us/library/hh847734.aspx

Scope

Variables have the scope in which they are created. You can change the scope of a variable with a scope prefix.

```
$[<scope-modifier>]:<name> = <value>

$global:computers = "Server01"
```

(Microsoft, 2015. Core Modules in Windows PowerShell) about_Variables, https://technet.microsoft.com/en-us/library/hh847734.aspx

Functions can have scope prefixes. This applies only to the scope of the function, not the variables within the function (for example local variables insides a global function are not available in the global scope).

```
function [<scope-modifier>]:<name> {<function-body>}
```

(Microsoft, 2015. Core Modules in Windows PowerShell) about_Scope, https://technet.microsoft.com/en-us/library/hh847849.aspx

Scope prefixes are three:

- global
- script
- private

(Microsoft, 2015. Core Modules in Windows PowerShell) about_Scope, https://technet.microsoft.com/en-us/library/hh847849.aspx

A child scope does not inherit the variables, aliases, and functions from the parent scope. Unless an item is private, the child scope can view the items in the parent scope. By default local variables can't alter the value of the parent variable as it is in the parent scope.

```
#
# Normal rules of scope inheritance
#
$mX = 10; # implicit script scope

function Dump-Variables {
  "`$global:mX $global:mX; `$script:mX $script:mX; `$mX $mx"
}

function Do-Something {
    "B: {0}" -f (Dump-Variables)
    $mX = 5; # Variable value changed only for local scope.
    "C: {0}" -f (Dump-Variables)
}

"A: {0}" -f (Dump-Variables)
Do-Something
"D: {0}" -f (Dump-Variables)

PS> .\Demo-Scope.ps1
A: $global:mX ; $script:mX 10; $mX 10
B: $global:mX ; $script:mX 10; $mX 10
C: $global:mX ; $script:mX 10; $mX 5
D: $global:mX ; $script:mX 10; $mX 10
```

(Microsoft, 2015. Core Modules in Windows PowerShell) about_Scope, https://technet.microsoft.com/en-us/library/hh847849.aspx

There is a difference between declaring a variable with a scope prefix and referencing a variable with a scope prefix.

```
# Declare variable with a scope prefix.
$global:mX = 10;

function global:Do-Something {
    Write-Host "B: $mX inherited value in child scope";
    $mX = 5;

# Reference variable with a scope prefix.
$mX = 10;

function global:Do-Something {
    Write-Host "B: $mX inherited value in child scope";
    $global:mX = 5;
```

(Microsoft, 2015. Core Modules in Windows PowerShell) about_Scope, https://technet.microsoft.com/en-us/library/hh847849.aspx

To alter the value of the parent variable as it is in the parent scope, reference the parent variable with a parent scope prefix.

```
$mX = 10; # implicit script scope

function Dump-Variables {
  "`$global:mX $global:mX; `$script:mX $script:mX; `$mX $mx"
}

function Do-Something {
    "B: {0}" -f (Dump-Variables)
    $script:mX = 5; # Alters the value in the script scope (and therefore the child local
scope)
    "C: {0}" -f (Dump-Variables)
}

"A: {0}" -f (Dump-Variables)
Do-Something
"D: {0}" -f (Dump-Variables)

PS> .\Demo-Scope.ps1
A: $global:mX ; $script:mX 10; $mX 10
B: $global:mX ; $script:mX 10; $mX 10
C: $global:mX ; $script:mX 5; $mX 5
D: $global:mX ; $script:mX 5; $mX 5
```

(Bentley, n.d.) 2015-08-21

A list of currently available variables.

Return a list of the currently available (to the session) variables using one of two techniques ...

```
# Technique 1
Get-Variable

# Technique 2
Set-Location variable: ; Get-ChildItem
```

(Microsoft, 2015. Core Modules in Windows PowerShell) about_Variables, https://technet.microsoft.com/en-us/library/hh847734.aspx

Automatic Variables

Automatic variables are those created and maintained by Powershell. They generally store information about the Windows and Powershell environment.

```
# Some Automatic Variables
$$
   Contains the last token in the last line received by the session.


$?
   Contains the execution status of the last operation. It contains
   TRUE if the last operation succeeded and FALSE if it failed.

$^
   Contains the first token in the last line received by the session.

$_
   Same as $PSItem. Contains the current object in the pipeline object.
   You can use this variable in commands that perform an action on every
   object or on selected objects in a pipeline.

$Args
   Contains an array of the undeclared parameters and/or parameter
   values that are passed to a function, script, or script block.
   When you create a function, you can declare the parameters by using the
   param keyword or by adding a comma-separated list of parameters in
   parentheses after the function name.

$False
```

```
   Contains FALSE. You can use this variable to represent
   FALSE in commands and scripts instead of using the string "false".
   The string can be interpreted as TRUE if it is converted to a non-empty
   string or to a non-zero integer.

$ForEach
   Contains the enumerator (not the resulting values) of a ForEach loop.
   You can use the properties and methods of enumerators on the value of the
   $ForEach variable. This variable exists only while the ForEach loop is running;
   it is deleted after the loop is completed. For detailed information, see
   about_ForEach.

$Host
    Contains an object that represents the current host application
    for Windows PowerShell. You can use this variable to represent the
    current host in commands or to display or change the properties of
    the host, such as $Host.version or $Host.CurrentCulture, or
    $host.ui.rawui.setbackgroundcolor("Red").

$Matches
   The $Matches variable works with the -match and -notmatch operators.
   When you submit scalar input to the -match or -notmatch operator, and
   either one detects a match, they return a Boolean value and populate
   the $Matches automatic variable with a hash table of any string values
   that were matched. For more information about the -match operator, see
   about comparison operators.

$NULL
   $null is an automatic variable that contains a NULL or empty value. You
   can use this variable to represent an absent or undefined value in commands
   and scripts.

$PSScriptRoot
    Contains the directory from which a script is being run.

$PsVersionTable
     Contains a read-only hash table that displays details about the
     version of Windows PowerShell that is running in the current session.

$Pwd
    Contains a path object that represents the full path of the current
    directory.

$StackTrace
    Contains a stack trace for the most recent error.

$This
    In a script block that defines a script property or script method, the
    $This variable refers to the object that is being extended.

$True
   Contains TRUE. You can use this variable to represent
   TRUE in commands and scripts.
```

(Microsoft, 2015. Core Modules in Windows PowerShell) get-help about_Automatic_Variables

Operators

Common

Type	Symbols
Arithmetic	+, -, *, /, %
Assignment	=, +=, -=, *=, /=, %=
Comparison	-eq, -ne, -gt, -lt, -le, -ge
Logical	-and, -or, -xor, -not, !
Unary	$x++ $x--

Type	-is, -isnot, -as
Bitwise	-bAND, -bOR, -bXOR, -bNOT

(Microsoft, 2015. Core Modules in Windows PowerShell) about_Operators, https://technet.microsoft.com/en-us/library/hh847732.aspx

String Operators

Type	**Symbols**
Concatenation	+
Containment	-in, -notin, -contains
Wildcard	-like, -notlike
Regex (regular expression)	-match, -notmatch, -replace
Split and Join	-split, -join

(Microsoft, 2015. Core Modules in Windows PowerShell) about_Operators, https://technet.microsoft.com/en-us/library/hh847732.aspx

Format	-f	Applies String.Format formatting. Enter the format string on the left side of the operator and the objects to be formatted on the right side of the operator.

Powershell Specific Types that aren't "Special"

Type	**Symbols**
Redirection	>, >>, 2>, 2>, and 2>&1

(Microsoft, 2015. Core Modules in Windows PowerShell) about_Operators, https://technet.microsoft.com/en-us/library/hh847732.aspx

"Special"

Arrays and Collections

Comma	,	As a binary operator, the comma creates an array. As a unary operator, the comma creates an array with one member.
Array subexpression	@( )	Returns the result of one or more statements as an array. If there is only one item, the array has only one member.
Range	..	Represents the sequential integers in an integer array, given an upper and lower boundary.
Index	[ ]	Selects objects from indexed collections, such as arrays and hash tables. Array indexes are zero-based, so the first object is indexed as [0]. For arrays (only), you can also use negative indexes to get the last values. Hash tables are indexed by key value.
Pipeline	|	Sends ("pipes") the output of the command that precedes it to the command that follows it. When the output includes more than one object (a "collection"), the pipeline operator sends the objects one at a time.

(Microsoft, 2015. Core Modules in Windows PowerShell) about_Operators, https://technet.microsoft.com/en-us/library/hh847732.aspx

Statement Invocation

Call	&	Lets you run commands that are stored in variables and represented by strings. Because the call operator does not parse the command, it cannot interpret command parameters.
Subexpression	$( )	Returns the result of one or more statements. For a single result, returns a scalar. For multiple results, returns an array. JLB the dollar prefix, $, is needed for creatiung subexpressions in strings.
JLB-Subexpression	( )	JLB - In practice it seems you can drop the $ and use plain brackets, ( ), for a subexpression.
Dot sourcing	.[Space]	Runs a script so that functions, aliases, and variables the script creates become available to the current scope.

(Microsoft, 2015. Core Modules in Windows PowerShell) about_Operators, https://technet.microsoft.com/en-us/library/hh847732.aspx

Objects

Member Reference	.	Accesses the properties and methods of an object.
Static Member Reference	::	
Cast	[ ]	Convert to the specified type.

(Microsoft, 2015. Core Modules in Windows PowerShell) about_Operators, https://technet.microsoft.com/en-us/library/hh847732.aspx

The Call Operator

Script blocks and the Call Operator

Assign a script block to a variable either, such that:

- The variable will require the call operator to invoke; or
- The variable will not require the call operator to invoke.

```
# The variable will require the call operator to invoke
PS> $a = { Set-Location variable:; Get-ChildItem }
PS> $a        # Just returns the script block
PS> &$a       # Invokes the script block

# The variable will not require the call operator to invoke.
PS> $a = &{ Set-Location variable:; Get-ChildItem }
PS> $a        # Invokes the script block.
```

(Microsoft, 2015. Core Modules in Windows PowerShell) about_Script_Blocks, https://technet.microsoft.com/en-us/library/hh847893.aspx

Functions and the Call operator

Assign a program or command that is invoked upon function run like this...

```
# Create
PS> function mlo { &("C:\Program Files (x86)\MyLifeOrganized.net\MLO\mlo.exe") }

# Use
PS> mlo
```

(Microsoft, 2014. Scripting with Windows PowerShell) Running Windows Programs, https://technet.microsoft.com/en-us/library/bb648600%28v=vs.85%29.aspx

Pipeline Operations

When you pipeline commands you pass objects, not text, between the commands.

```
PS> Get-ChildItem | Format-Table -AutoSize
```

(Microsoft, 2014. Scripting with Windows PowerShell) Object Pipeline, https://technet.microsoft.com/en-us/library/dd347655.aspx

Object Operations

Where-Object (where)

The Where-Object cmdlet, aliased as "where", returns a subset of the collection passed to it.

(Microsoft, 2015. Core Modules in Windows PowerShell) Where-Object, https://technet.microsoft.com/en-us/library/hh849715.aspx

Where-Object syntax.

```
# Script block syntax
<command> | (Where-Object|where) { $ .<property> <comparision operator> <value> }

# Statement syntax, explicit parameter names.
<command> | (Where-Object|where) -Property <property> <comparison operator> -Value <value>

# Statement syntax, implicit parameter names.
<command> | (Where-Object|where) <property> <comparison operator> <value>
```

```
# Script block syntax
PS> Get-ChildItem | where { $_.Length -LT 100 }

# Statement syntax, explicit parameter names.
PS> Get-ChildItem | where -Property Length -LT -Value 100

# Statement syntax, implicit parameter names. (The following are equivalent).
PS> Get-ChildItem | where length -LT 100
PS> Get-ChildItem | Where-Object length -LT 100
```

(Microsoft, 2015. Core Modules in Windows PowerShell) Where-Object, https://technet.microsoft.com/en-us/library/hh849715.aspx

ForEach-Object (foreach)

ForEach is useful for iterating over all items in a collection, including arrays.

(Microsoft, 2015. Core Modules in Windows PowerShell) about_ForEach, https://technet.microsoft.com/en-us/library/hh847816.aspx

In a pipeline context a foreach statement is technically a ForEach-Object cmdlet. "foreach" is an alias for the ForEach-Object cmdlet.

(Microsoft, 2015. Core Modules in Windows PowerShell) ForEach-Object, https://technet.microsoft.com/en-us/library/hh849731.aspx

ForEach syntax.

```
# Script syntax
(foreach|ForEach-Object) ($<item> in $<collection>){<statement list>}

# Pipeline syntax
<command> | (foreach|ForEach-Object) {<beginning command_block>}{<middle
command_block>}{<ending command_block>}

# Pipeline syntax, automatic variable use, explicit.
<command> | (foreach|ForEach-Object) {$_.<object member>}

# Pipeline syntax, automatic variable use, implicit.
<command> | (foreach|ForEach-Object) <object member>
```

```
# Script
foreach ($item in Get-ChildItem)
{
    $item.Name
    $item.IsReadOnly # Only applies to files, not directories.
    $item.LastAccessTime
    "`n"
}

# Pipeline
Get-ChildItem | foreach { 'beginning' } { $ .Name } { 'ending' }

# Pipeline syntax, automatic variable use, explicit.
Get-ChildItem | foreach  { $_.Name }

# Pipeline syntax, automatic variable use, implicit
Get-ChildItem | foreach  Name
```

(Microsoft, 2015. Core Modules in Windows PowerShell) about_ForEach, https://technet.microsoft.com/en-us/library/hh847816.aspx

To return the value of the object instance's property, just directly refer to the property name. ...

```
# Get object property value
PS> Get-ChildItem | ForEach-Object BaseName
countries
debug
file01
file02
file03
file04
LICENSE
```

(Bentley, n.d.) 2017-02-24 18:00

To return the value of the object instance's method, use:
* ForEach-Object;
* the item syntax ($_.Property or $_.Method()); and
* Surrounded by brackets "{}" to evaluate as an expression.

```
# Get Method Return Values
PS> Get-Content countries.txt | ForEach-Object {$_.ToString()}

Afghanistan
Albania
Algeria
Andorra
```

(Bentley, n.d.) 2017-02-24 18:00

The following truncates string output.

```
ps> Get-Content countries.txt | ForEach-Object {$_.Substring(0,4)}
Afgh
Alba
Alge
Ando

ps> Get-ChildItem | ForEach-Object {$_.Name.Substring(0,4)}
.git
coun
debu
file

ps> Get-Content countries.txt | ForEach-Object {$_[0..3] -join ""}
Afgh
Alba
Alge
And

ps> Get-ChildItem | Format-Table @{Expression={$_.Name};Label="FileName";width=4}

File
Name
----
....
c...
d...
f...
```

(Bentley, n.d.) 2017-02-19 16:12

Sort-Object (sort)

Use `Sort-Object` to sort.

```
PS> Get-ChildItem -Recurse | Sort-Object -Property Length -Descending |
 Format-Table -AutoSize -GroupBy 'FakePropertyToSupressGroupBy'
```

(Microsoft, 2014. Scripting with Windows PowerShell) Sorting Objects, https://technet.microsoft.com/en-us/library/dd347718.aspx

Select-Object (select)

Select-Object allows you to create a new object (of type System.Management.Automation.PSCustomObject) using a piped-in object with some subset of its properties. One of the chief uses in creating a PSCustomObject is that it enables you to modify it's properties.

```
# Create a new object using Select-Object
PS> Get-ChildItem | Select-Object -Property Name, Length | Format-Table -AutoSize

Name                            Length
----                            ------
dir01
dir02
Adb-Connect.ps1 - Shortcut.lnk    1293
Demo-Arrays.ps1                   2409
Demo-CommentBasedHelp.ps1         1831
Demo-ControlFlow.ps1              1013
...

# Create a new object, create a new property and modify an existing property
PS> Get-ChildItem `
| Select-Object -Property Name, Length, CustomPropLength `
```

```
| ForEach-Object { $_.CustomPropLength = $_.Length * 100; $_.Length = 4; $_ } `
| Format-Table -AutoSize

Name                              Length CustomPropLength
----                              ------ ----------------
dir01                                  4
dir02                                  4
Adb-Connect.ps1 - Shortcut.lnk         4 129300
Demo-Arrays.ps1                        4 240900
Demo-CommentBasedHelp.ps1              4 183100
Demo-ControlFlow.ps1                   4 101300
Demo-Functions.ps1                     4 128500
Demo-Hashes.ps1                        4 117100
```

(Microsoft, 2014. Scripting with Windows PowerShell) Selecting Parts of Objects (Select-Object), https://technet.microsoft.com/en-us/library/dd347697.aspx
(Bentley, n.d.) 2015-07-31 12:17

Use Select-Object to limit output to the first N lines.

```
ps> Get-Content countries.txt | Select-Object -First 3
Afghanistan
Albania
Algeria
```

(Bentley, n.d.) 2017-02-19 16:05

Use Select Object to output the value of an objects property.

```
PS> Invoke-WebRequest http://twf.org.au/research/CatchingLightning.html


StatusCode        : 200
StatusDescription : OK
Content           : <!DOCTYPE HTML PUBLIC "-//W3C//DTD HTML 3.2//EN">
                    <html>

                    <head>
...

PS> Invoke-WebRequest http://twf.org.au/research/CatchingLightning.html | Select-Object -
ExpandProperty Content
<!DOCTYPE HTML PUBLIC "-//W3C//DTD HTML 3.2//EN">
<html>

<head>
<title>Catching Lightning in a Bucket: Archiving the Performing Arts in the Digital
Age</title>
```

https://stackoverflow.com/questions/14406315/how-to-get-an-objects-propertys-value-by-property-name
(Bentley, n.d.) 2019-08-21

Use Select Objects to remove duplicate lines from output.

```
PS> C:\Users\John> Get-Process -Name svc* | Format-Table -Property Name

Name
----
svchost
svchost
svchost
svchost

PS C:\Users\John> Get-Process -Name svc* | Select-Object -Unique | Format-Table -Property
Name

Name
----
svchost
```

Working with Objects

.Net

The reference for .net objects: (Microsoft, 2015..NET Framework Class Library)
https://msdn.microsoft.com/en-us/library/gg145045%28v=vs.110%29.aspx

Create a new .Net object instance with `New-Object`. Assign that object to a variable and you can subsequently reference the properties and methods of that .Net object instance.

```
$sb = New-Object -TypeName System.Text.StringBuilder
Get-ChildItem | foreach { $sb.AppendFormat("{0}`r`n", $_.Name) } | Out-Null
$sb.ToString()
```

Powershell automatically prepends "System" to type names when you use New-Object, but it is better to be explicit.

(Microsoft, 2014. Scripting with Windows PowerShell) Creating .NET and COM Objects (New-Object),
https://technet.microsoft.com/en-us/library/dd347574.aspx
(Bentley, n.d.) 2015-07-27 10:25

To reference a .Net class itself, whether a static class or not, enclose the class in brackets.

```
PS> [System.Text.StringBuilder]

IsPublic IsSerial Name                                     BaseType
-------- -------- ----                                     --------
True     True     StringBuilder                            System.Object

PS> [System.Text.StringBuilder] | Get-Member


   TypeName: System.RuntimeType

Name                          MemberType Definition
----                          ---------- ----------
AsType                        Method     type AsType()
Clone                         Method     System.Object Clone(), System.Obje...
Equals                        Method     bool Equals(System.Object obj), bo...
FindInterfaces                Method     type[] FindInterfaces(System.Refle...
FindMembers                   Method     System.Reflection.MemberInfo[] Fin..
...


PS> [System.Environment]

IsPublic IsSerial Name                                     BaseType
-------- -------- ----                                     --------
True     False    Environment                              System.Object

PS> [System.Environment] | Get-Member


   TypeName: System.RuntimeType

Name                          MemberType Definition
----                          ---------- ----------
AsType                        Method     type AsType()
Clone                         Method     System.Object Clone(), System.Obje...
Equals                        Method     bool Equals(System.Object obj), bo...
FindInterfaces                Method     type[] FindInterfaces(System.Refle...
```

(Microsoft, 2014. Scripting with Windows PowerShell) Using Static Classes and Methods,
https://technet.microsoft.com/en-us/library/dd347632.aspx

To discover the static members of a class use the -Static parameter of Get-Member.

```
PS> [System.Environment] | Get-Member -Static


   TypeName: System.Environment

Name                        MemberType Definition
----                        ---------- ----------
Equals                      Method     static bool Equals(System.Object objA,...
Exit                        Method     static void Exit(int exitCode)
ExpandEnvironmentVariables Method     static string ExpandEnvironmentVariabl...
FailFast                    Method     static void FailFast(string message), ...
...
CommandLine                 Property   static string CommandLine {get;}
CurrentDirectory            Property   static string CurrentDirectory {get;set;}
CurrentManagedThreadId      Property   static int CurrentManagedThreadId {get;}
ExitCode                    Property   static int ExitCode {get;set;}
...
```

(Microsoft, 2014. Scripting with Windows PowerShell) Using Static Classes and Methods, https://technet.microsoft.com/en-us/library/dd347632.aspx

To use the static members of a class use a brackets and two colon syntax...

```
[<Qualfied.Class.Name>]::<Member>
```

```
# Access a static property
PS> [System.Environment]::OSVersion | Format-Table -AutoSize

Platform ServicePack Version    VersionString
-------- ----------- -------    -------------
 Win32NT             6.2.9200.0 Microsoft Windows NT 6.2.9200.0

# Use a static method
PS> [System.Math]::Sqrt(9)
3
```

(Microsoft, 2014. Scripting with Windows PowerShell) Using Static Classes and Methods, https://technet.microsoft.com/en-us/library/dd347632.aspx

COM

Create a new COM object with `New-Object -ComObject`.

```
New-Object -ComObject WScript.Shell
New-Object -ComObject WScript.Network
New-Object -ComObject Scripting.Dictionary
New-Object -ComObject Scripting.FileSystemObject
```

(Microsoft, 2014. Scripting with Windows PowerShell) Creating .NET and COM Objects (New-Object), https://technet.microsoft.com/en-us/library/dd347574.aspx

Working with a COM object example: Use Windows Script Host to create a Windows Shortcut.

```
$WshShell = New-Object -ComObject WScript.Shell
$shortcut = $WshShell.CreateShortcut("$Home\Documents\zTemp\TempShortcut.lnk")
$shortcut.TargetPath = "$Home\Documents"
$shortcut.Save()

# You can't resolve COM properties within double quotes
"ShortCut created: " +  $shortcut.FullName

$shortcut | Get-Member
```

(Microsoft, 2014. Scripting with Windows PowerShell) Creating .NET and COM Objects (New-Object), https://technet.microsoft.com/en-us/library/dd347574.aspx

Arrays

Create and Initialise

Create and initialise, comma and range operator syntax.

```
# Create and initialise: comma syntax (from strings)
$b = "Fun", "Times", "Turkey"
$b.GetType() # Object[]

# Create and initialise: comma syntax (from numbers)
$a = 34, 56, 67
$b.GetType() # Object[]

# Create and initialise: range operator syntax
$c = 10..20
$c.GetType() # Object[]
```

(Microsoft, 2015. Core Modules in Windows PowerShell) about_Arrays, https://technet.microsoft.com/en-us/library/hh847882.aspx
C:\Users\John\Documents\Sda\Code\Windows\PowerShell\Examples\Demo-Arrays.ps1

Create and initialise, strongly typed syntax.

```
# Create and initialise: Strongly typed (string)
[string[]]$d = "Far", "from", "Over"
$d
$d.GetType() # String[]

# Create and initialise: Strongly typed (numbers)
[int[]]$e = 5, 14, 20
$e.GetType() # Int32[]

# Create and initialise: Strongly typed (.Net object)
[Diagnostics.Process[]]$f = Get-Process
$f.GetType() # Process[]
```

(Microsoft, 2015. Core Modules in Windows PowerShell) about_Arrays, https://technet.microsoft.com/en-us/library/hh847882.aspx
C:\Users\John\Documents\Sda\Code\Windows\PowerShell\Examples\Demo-Arrays.ps1

Create and initialise, sub-expression operator. The main use for a sub-expression operator is in creating an arrary from one or zero objects. Useful when you don't know how many objects will be returned.

```
# Create and initialise: sub-expression operator (Many objects)
$g = @("Now", 14, "Cool")

# Create and initialise: sub-expression operator (one object)"
$h = @("Queen")

# Create and initialise: sub-expression operator (no objects)"
$k = @()

# Useful when you don't know how many objects will be returned.
$p = @(Get-Process Notepad)
```

(Microsoft, 2015. Core Modules in Windows PowerShell) about_Arrays, https://technet.microsoft.com/en-us/library/hh847882.aspx
C:\Users\John\Documents\Sda\Code\Windows\PowerShell\Examples\Demo-Arrays.ps1

Read

Read arrays.

```
# The array.
$a = "beauty", 'fun', "cool", 12, 34, "nice", "zara", "penny", "nicki"

# Return all array
$a

# Return an element
$a[2]

# Return a range of elements
$a[2..4]

# Retun last three elements
$a[-3..-1]

# Combine a list with a range with the + operator
$a[1,2+4..6]
```

(Microsoft, 2015. Core Modules in Windows PowerShell) about_Arrays, https://technet.microsoft.com/en-us/library/hh847882.aspx
C:\Users\John\Documents\Sda\Code\Windows\PowerShell\Examples\Demo-Arrays.ps1

Count of array elements.

```
$a.Count
```

(Microsoft, 2015. Core Modules in Windows PowerShell) about_Arrays, https://technet.microsoft.com/en-us/library/hh847882.aspx

Use in iterations.

```
foreach ($element in $a) {$element}

for ($i = 0; $i -le ($a.length - 1); $i += 2) {$a[$i]}

$i=0
while($i -lt 4) {$a[$i]; $i++}
```

(Microsoft, 2015. Core Modules in Windows PowerShell) about_Arrays, https://technet.microsoft.com/en-us/library/hh847882.aspx

Get Information

Fetch the members, the properties and methods, of the array class.

```
$x = @()
Get-Member -InputObject $x

#Output
Name            MemberType            Definition
----            ----------            ----------
Count           AliasProperty         Count = Length
Add             Method                int IList.Add(System.Object value)
Address         Method                System.Object&, mscorlib, Version=4.0.0.0,
Culture=neutral, Public...
Clear           Method                void IList.Clear()
...
```

(Microsoft, 2015. Core Modules in Windows PowerShell) about_Arrays, https://technet.microsoft.com/en-us/library/hh847882.aspx
C:\Users\John\Documents\Sda\Code\Windows\PowerShell\Examples\Demo-Arrays.ps1

Get the datatype of each element of the array, duplicates eliminated.

```
$a = "beauty", 'fun', "cool", 12, 34, "nice", "zara", "penny", "nicki"
$a | Get-Member

# Output
   TypeName: System.String

Name             MemberType            Definition
----             ----------            ----------
Clone            Method                System.Object Clone(), System.Object
ICloneable.Clone()
CompareTo        Method                int CompareTo(System.Object value), int
CompareTo(string strB),...

   TypeName: System.Int32

Name        MemberType Definition
----        ---------- ----------
CompareTo   Method     int CompareTo(System.Object value), int CompareTo(int value), int
IComparable.Co...
Equals      Method     bool Equals(System.Object obj), bool Equals(int obj), bool
IEquatable[int].Equal...
GetHashCode Method     int GetHashCode()
...
```

(Microsoft, 2015. Core Modules in Windows PowerShell) about_Arrays, https://technet.microsoft.com/en-us/library/hh847882.aspx
C:\Users\John\Documents\Sda\Code\Windows\PowerShell\Examples\Demo-Arrays.ps1

Array manipulation

Set an element.

```
$a[2] = "fresh"
$a.SetValue(500,1) # The 1 is the element index, 500 the value.
```

(Microsoft, 2015. Core Modules in Windows PowerShell) about_Arrays, https://technet.microsoft.com/en-us/library/hh847882.aspx
C:\Users\John\Documents\Sda\Code\Windows\PowerShell\Examples\Demo-Arrays.ps1

Add an element.

```
$a += "Tool" # Internally creates a new array with the element added.
```

(Microsoft, 2015. Core Modules in Windows PowerShell) about_Arrays, https://technet.microsoft.com/en-us/library/hh847882.aspx
C:\Users\John\Documents\Sda\Code\Windows\PowerShell\Examples\Demo-Arrays.ps1

Remove an element.

```
# Remove an element (at index 3, the 4th element)"
$t = $a[0,1,2 + 4..($a.Length - 1)]
```

(Microsoft, 2015. Core Modules in Windows PowerShell) about_Arrays, https://technet.microsoft.com/en-us/library/hh847882.aspx
C:\Users\John\Documents\Sda\Code\Windows\PowerShell\Examples\Demo-Arrays.ps1

Delete the array.

```
$a = $null
```

(Microsoft, 2015. Core Modules in Windows PowerShell) about_Arrays, https://technet.microsoft.com/en-us/library/hh847882.aspx
C:\Users\John\Documents\Sda\Code\Windows\PowerShell\Examples\Demo-Arrays.ps1

Combine two arrays.

```
$z = $x + $y
```

(Microsoft, 2015. Core Modules in Windows PowerShell) about_Arrays, https://technet.microsoft.com/en-us/library/hh847882.aspx
C:\Users\John\Documents\Sda\Code\Windows\PowerShell\Examples\Demo-Arrays.ps1

Datetimes

Basics.

```
PS> [datetime]$mydate = "24Jan2020"
PS> $mydate

Friday, 24 January 2020 00:00:00

PS> $mydate | Get-Member

   TypeName: System.DateTime

Name                 MemberType     Definition
----                 ----------     ----------
Add                  Method         datetime Add(timespan value)
AddDays              Method         datetime AddDays(double value)
...
ToString             Method         string ToString(), string ToString(string format),...
...
Date                 Property       datetime Date {get;}
Day                  Property       int Day {get;}
DayOfWeek            Property       System.DayOfWeek DayOfWeek {get;}
DayOfYear            Property       int DayOfYear {get;}
Hour                 Property       int Hour {get;}
...

PS> $mydate.toString("yyyy-MM-dd")
2020-01-24
```

Get the current date and time.

```
> Get-Date -Format yyyyMMdd-hhmm
20190901-0748

# ISO8601
> Get-Date -Format "yyyy-MM-dd HH:mm"
2021-05-19 06:44
```

https://www.sconstantinou.com/powershell-date-format/

Hash Tables

Basics

Hash tables store key/value pairs. Also known as "dictionary" or "associative array". Hash tables can be optionally ordered: keys are maintained in the order you list them (not necessarily alphabetic or number order). Keys and values can be of any object type, although they are often strings and integers.

(Microsoft, 2015. Core Modules in Windows PowerShell) about_Hash_Tables, https://technet.microsoft.com/en-us/library/hh847780.aspx

Hash table syntax.

```
[ordered]@{ [<name> = <value>;] [<name> = <value>] ...}

- A Key that contains spaces must be enclosed in quotation marks. Values must be valid
Windows PowerShell expressions. Strings must appear in quotation marks, even if they do not
include spaces.
```

```
# Create and initialize hash table
$x = @{ Score = 10; Team = "Blues"; Cup = "The Alpine" ; "Fonzi Winkler" = "Happy Days"}

# Create an empty hash table
$x = @{}

# Create an ordered dictionary.
y = [ordered]@{ Score = 10; Team = "Blues"; Cup = "The Alpine" }
```

(Microsoft, 2015. Core Modules in Windows PowerShell) about_Hash_Tables, https://technet.microsoft.com/en-us/library/hh847780.aspx

Keys and Values

You can return just the keys or values of hash table (includind an ordered dictionary).

```
$x.Keys

$x.Values
```

(Microsoft, 2015. Core Modules in Windows PowerShell) about_Hash_Tables, https://technet.microsoft.com/en-us/library/hh847780.aspx

Each keyname gets stored as a property of the hash table.

```
$x.Score

#Returns
10
```

(Microsoft, 2015. Core Modules in Windows PowerShell) about_Hash_Tables, https://technet.microsoft.com/en-us/library/hh847780.aspx

Add a key/value pair.

```
# Any of the following works either to add or update a key/value pair.
$hash["Force"] = 10  # Keys must be surrounded by quotes.
$hash['Clear'] = 5

# Doesn't work (official documentation wrong)
$hash = $hash.Add("Force", 20)

# OK.
$hash.Add("Force", 10)

# OK.
$hash = $hash + @{ "Family" = 4 }
```

(Microsoft, 2015. Core Modules in Windows PowerShell) about_Hash_Tables, https://technet.microsoft.com/en-us/library/hh847780.aspx
(Bentley, n.d.) 2015-07-14 07:37

You can add a key/value pair from variables.

```
$key = "Bravado"
$value = "mild"
```

```
$hash.Add($key, $value)
```

(Microsoft, 2015. Core Modules in Windows PowerShell) about_Hash_Tables, https://technet.microsoft.com/en-us/library/hh847780.aspx
(Bentley, n.d.) 2015-07-14 07:39

Update a key/value pair

```
# Any of the following works either to add or update a key/value pair.
$hash["Force"] = 10  # Keys must be surrounded by quotes.
$hash['Clear'] = 5
```

(Bentley, n.d.) 2021-01-27 00:18 Demo-Hashes.ps1

Remove a key/value pair.

```
$hash.Remove("Team")
```

(Microsoft, 2015. Core Modules in Windows PowerShell) about_Hash_Tables, https://technet.microsoft.com/en-us/library/hh847780.aspx
(Bentley, n.d.) 2015-07-14 07:43

The hash object has other properties and methods for you to use. Search for "MSDN System.Collections.Hashtable"

Store any Object type

You can store any .net object in the key or value of a hash table.

```
$p = @{'PowerShell' = (Get-Process PowerShell); `
    "Notepad" = (get-process notepad); `
    (Get-Service WinRM) = ((Get-Service WinRM).Status) }

$p

#Result
Name                           Value
----                           -----
PowerShell                     System.Diagnostics.Process (powershell)
Notepad                        System.Diagnostics.Process (notepad)
WinRM                          Stopped

$p.Keys

#Result
PowerShell
Notepad

Status   Name               DisplayName
------   ----               -----------
Stopped  WinRM              Windows Remote Management (WS-Manag...
```

(Microsoft, 2015. Core Modules in Windows PowerShell) about_Hash_Tables, https://technet.microsoft.com/en-us/library/hh847780.aspx
(Bentley, n.d.) 2015-07-14 14:23
(Microsoft, 2015..NET Framework Class Library) https://msdn.microsoft.com/en-us/library/gg145045%28v=vs.110%29.aspx

You can have a hash assigned to one of a hash's values.

```
$hash.Add("Hash2", @{a=1; b=2; c=3})
$hash

Name                           Value
```

```
----                           -----
Score                          10
Cup                            The Alpine
Fonzi Winkler                  Happy Days
Force                          10
Bravado                        mild
Family                         4
Hash2                          {c, b, a}

$hash.Hash2
Name                           Value
----                           -----
c                              3
b                              2
a                              1

$hash.Hash2.b
2
```

(Microsoft, 2015. Core Modules in Windows PowerShell) about_Hash_Tables, https://technet.microsoft.com/en-us/library/hh847780.aspx
(Bentley, n.d.) 2015-07-14 14:23

Get sorted data

You can't sort a hash table but you can get sorted results from it.

```
$hash.GetEnumerator() | Sort-Object -Property Key -Descending

Name                           Value
----                           -----
Score                          10
Hash2                          {c, b, a}
Force                          10
Fonzi Winkler                  Happy Days
Family                         4
Cup                            The Alpine
Bravado                        mild
```

(Microsoft, 2015. Core Modules in Windows PowerShell) about_Hash_Tables, https://technet.microsoft.com/en-us/library/hh847780.aspx
(Bentley, n.d.) 2015-07-14 14:44

ConvertFrom-StringData

You can convert a string or a here string into a hash table with ConvertFrom-StringData.

```
$myString = @"
Nice = Funny "Stuff".
Tough = But oh so "Cool."
Rough = Because it's nice.
"@

$convertedHash = ConvertFrom-StringData $myString
$convertedHash

# Result
Name                           Value
----                           -----
Tough                          But oh so "Cool."
Nice                           Funny "Stuff".
Rough                          Because it's nice.
```

(Microsoft, 2015. Core Modules in Windows PowerShell) about_Hash_Tables, https://technet.microsoft.com/en-us/library/hh847780.aspx
(Bentley, n.d.) 2015-07-14 14:44

Splatting

Splatting is a method of passing a collection of parameter values to a command as a unit, using a variable. When you pass the variable to a command you use the at symbol (@) rather than the normal dollar ($) sign.

```
# Conventionally
Copy-Item -Path "temp1.txt" -Destination "temp2.txt" -WhatIf

# Assign an array or hash table of parametername, parametervalue pairs to a variable. The
syntax for hash tables in Windows PowerShell is: @{ <name>=<value>; <name>=<value>; …}. The
'@' here is for the hashtable.
$hashArguments = @{ Path = "temp1.txt"; Destination = "temp2.txt"; WhatIf = $true}

# Invoke the command using splatting. The '@' here serves as the splatting operator.
Copy-Item @hashArguments
```

(Microsoft, 2015. Core Modules in Windows PowerShell) about_Splatting, https://technet.microsoft.com/en-us/library/jj672955.aspx

The variable passed to the command can be a Hash Table, suitable for named parameters.

```
# Conventionally
Copy-Item -Path "temp1.txt" -Destination "temp2.txt" -WhatIf

# Using splatting
$hashArguments = @{ Path = "temp1.txt"; Destination = "temp2.txt"; WhatIf = $true}
Copy-Item @hashArguments
```

(Microsoft, 2015. Core Modules in Windows PowerShell) about_Splatting, https://technet.microsoft.com/en-us/library/jj672955.aspx

The variable passed to the command can be an array, suitable for positional parameters.

```
# Conventional use
Copy-Item 'temp1.txt' 'temp2.txt' -WhatIf

# Using splatting
$arrayArguments = 'temp1.txt', 'temp2.txt'
Copy-Item @arrayArguments -WhatIf
```

Command Shortcuts

Basics

You can use and create shortcuts for commands, or a series of commands in two ways:

- Aliases.
- Functions.

The documentation claims you can't create aliases for commands that take parameters. But you can!

(Microsoft, 2014. Scripting with Windows PowerShell) Using Aliases, https://technet.microsoft.com/en-us/library/bb648603%28v=vs.85%29.aspx

Aliases

Get a list of all aliases.

```
# Technique 1
Get-Alias

# Technique 2
Set-Location alias:; Get-ChildItem
```

(Microsoft, 2014. Scripting with Windows PowerShell) Using Aliases, https://technet.microsoft.com/en-us/library/bb648603%28v=vs.85%29.aspx

Get all aliases for a command.

```
Get-Alias -Definition Get-ChildItem
```

(Bentley, n.d.) 2015-07-21 08:28

Create an alias.

```
Set-Alias gh Get-Help

# Use the alias with a parameter
gh Get-ChildItem
```

(Bentley, n.d.) 2015-07-21 08:32

Delete an alias.

```
Remove-Item alias:jlb*
```

(Bentley, n.d.) 2015-07-21 08:34

Functions

Creating functions in effect creates code that can be reused much like an alias.

```
# Create
function bootini {notepad c:\boot.ini}

# Use
bootini
```

(Microsoft, 2014. Scripting with Windows PowerShell) Using Aliases, https://technet.microsoft.com/en-us/library/bb648603%28v=vs.85%29.aspx

Drives and Providers

Traditionally "drives" are understood as file system drives, like "C:". Powershell extends the notion of drives, by analogy, to other parts of the operating system (E.g The certificate store, and registry, environment variables) and other aspects of Powershell (Available functions and Variables). This enables you to consistently work with parts of the operating system and powershell itself in the same manner that you would work with a file system drive.

(Microsoft, 2014. Scripting with Windows PowerShell) About Windows PowerShell Drives, https://technet.microsoft.com/en-us/library/bb613483%28v=vs.85%29.aspx

Get a list of drives.

```
Get-PSDrive | Format-Table -AutoSize

Name     Used (GB) Free (GB) Provider    Root
```

```
----     --------- --------- --------    ----
Alias                        Alias
C           314.17    122.44 FileSystem  C:\
Cert                         Certificate \
D           443.04     22.72 FileSystem  D:\
Env                          Environment
Function                     Function
HKCU                         Registry    HKEY_CURRENT_USER
HKLM                         Registry    HKEY_LOCAL_MACHINE
Variable                     Variable
WSMan                        WSMan
X                            FileSystem  X:\
```

(Bentley, n.d.) 2015-07-23 10:24

Change to a drive and list the drive's items.

```
// Show the Windows Environment Variables
PS> Set-Location Env:
PS Env:\> Get-ChildItem
Name                           Value
----                           -----
ALLUSERSPROFILE                C:\ProgramData
APPDATA                        C:\Users\John\AppData\Roaming
CommonProgramFiles             C:\Program Files\Common Files
CommonProgramFiles(x86)        C:\Program Files (x86)\Common Files
…
```

(Bentley, n.d.) 2017-11-27 12:11

Set and display drive item (for session only).

```
// If you are on the relevant drive. E.g. Set a Windows Environment Variable
PS Env:\> Set-Item -Path JLB -Value Cool
PS Env:\> Get-ChildItem -Path JLB
Name                           Value
----                           -----
JLB                            Fun

// Do so from another drive
PS> Set-Item -Path Env:JLB -Value Fun
PS> Get-ChildItem -Path Env:JLB
Name                           Value
----                           -----
JLB                            Fun

// Shortcut to Set
PS> $Env:JLB = "Dog"

// Shortcut to Display
PS> $Env:JLB
Dog
```

(Bentley, n.d.) 2017-11-27 12:17

Set and display drive item (permanently).

```
// Get a System level environment variable
PS> [Environment]::GetEnvironmentVariable("Path", "Machine")
PS> [Environment]::GetEnvironmentVariable("JLB", "Machine")

// Set a System level environment variable.
* Open Powershell as an Administrater
PS> [Environment]::SetEnvironmentVariable("JLB", "Foo", "Machine")
```

(Bentley, n.d.) 2017-11-27 12:33

Delete a drive item (permanently).

```
// Delete a System level environment variable.
* Open Powershell as an Administrater
PS> [Environment]::SetEnvironmentVariable("JLB", $null, "Machine")
```

Create a drive and use it.

```
PS> New-PSDrive -Name Fun -PSProvider FileSystem -Root "C:\Users\John\Documents\zTemp"
PS> Set-Location fun:
PS> Get-ChildItem
```

(Microsoft, 2014. Scripting with Windows PowerShell) About Windows PowerShell Drives, https://technet.microsoft.com/en-us/library/bb613483%28v=vs.85%29.aspx

Providers entail a drive type. So a single provider might make avaible many drive instances. For example, the FileSystem provider might make available the drives C, D, X. To list all the Providers ...

```
PS> Get-PSProvider | Format-Table -AutoSize

Name        Capabilities                       Drives
----        ------------                       ------
Alias       ShouldProcess                      {Alias}
Environment ShouldProcess                      {Env}
FileSystem  Filter, ShouldProcess, Credentials {C, D, Fun, X}
Function    ShouldProcess                      {Function}
Registry    ShouldProcess, Transactions        {HKLM, HKCU}
Variable    ShouldProcess                      {Variable}
Certificate ShouldProcess                      {Cert}
WSMan       Credentials                        {WSMan}
```

(Microsoft, 2014. Scripting with Windows PowerShell) Drives and Providers, https://technet.microsoft.com/en-us/library/bb648606%28v=vs.85%29.aspx

Extending Powershell

Basics

Powershell can be extended with Modules and Snap-Ins. Modules contain: cmdlets; providers; functions; aliases; variables; and drives. Snap-Ins contain: cmdlets; and providers only.

(Microsoft, 2014. Scripting with Windows PowerShell) Using Modules and Snap-Ins, https://technet.microsoft.com/en-us/library/dd745031%28v=vs.85%29.aspx
(Microsoft, 2015. Core Modules in Windows PowerShell) about_Modules, https://technet.microsoft.com/en-us/library/hh847804.aspx

For a module or snap-in you need to:

- Obtain it or create it yourself,
- Install it.
- Import it.
- Discover the commands.
- Use the commands.

Using

Modules

To install a module copy it to one of the directories specified by `$env:PSModulePath`. For example ...

```
Copy-Item -Path c:\ps-test\MyModule -Destination $home\Documents\WindowsPowerShell\Modules
```

(Microsoft, 2015. Core Modules in Windows PowerShell) about_Modules, https://technet.microsoft.com/en-us/library/hh847804.aspx

Importing a module is automatic if it is in one of the directories specified by `$env:PSModulePath`, and you run any command from the module for the first time.

(Microsoft, 2015. Core Modules in Windows PowerShell) about_Modules, https://technet.microsoft.com/en-us/library/hh847804.aspx

To import a module not in one of the directories specified by `$env:PSModulePath`.

```
# Put this in your profile if you want the module available for every session.
Import-Module c:\ps-test\TestCmdlets
```

(Microsoft, 2015. Core Modules in Windows PowerShell) about_Modules, https://technet.microsoft.com/en-us/library/hh847804.aspx

A list of all available modules:

```
Get-Module -ListAvailable
```

(Microsoft, 2015. Core Modules in Windows PowerShell) about_Modules, https://technet.microsoft.com/en-us/library/hh847804.aspx

A list of modules already imported into the session.

```
Get-Module
```

(Microsoft, 2015. Core Modules in Windows PowerShell) about_Modules, https://technet.microsoft.com/en-us/library/hh847804.aspx

Get help on a module.

```
Get-Help -Name Pscx
```

(Bentley, n.d.) 2015-07-21 07:18

Snap-ins

These are .NET assemblies delivered as .dlls.

(Microsoft, 2015. Core Modules in Windows PowerShell) about_PSSnapins, https://technet.microsoft.com/en-us/library/hh847795.aspx

Scripts

Structure

"Scripts" are a set of powershell statements saved to a *.ps1 file.

Scripts can handle a pipeline.

```
# Demo-ScriptPipeline.ps1
begin {
'Starting ...'
}
process {
"$_"
}
end {
'Ended'
}

# From the console.
PS> 1, 2, 3 | .\Demo-ScriptPipeline.ps1
Starting ...
1
2
3
Ended
```

(Bentley, n.d.) *\\ATLAS\Documents\Sda\Code\Windows\PowerShell\Examples\Demo-ScriptPipeline.ps1*

Pass arguments to a Script by position.

```
# Demo-ScriptArguments.ps1
foreach ($arg in $args) {
   Get-Item -Path $arg | Format-Table Name, LastAccessTime -AutoSize
}

# Console
PS> .\Demo-ScriptArguments.ps1 temp1.txt, temp2.txt

Name      LastAccessTime
----      --------------
temp1.txt 2015-07-Jul-14-Tue 14:49:35
temp2.txt 2015-07-Jul-20-Mon 09:02:02
```

(Bentley, n.d.) *\\ATLAS\Documents\Sda\Code\Windows\PowerShell\Examples\Demo-ScriptArguments.ps1*

Pass arguments to a script by name.

```
# Demo-ScriptArguments-ByNameParams.ps1
Param
(
  [parameter(Mandatory=$true)]
  [String]
  $BaseName,

  [parameter(Mandatory=$false)]
  [Int]
  $Factor
)
Write-Host $BaseName $Factor

# Console
PS> .\Demo-ScriptArguments-ByNameParams.ps1 -BaseName fun -Factor 12
fun 12
```

C:\Users\John\Documents\Sda\Code\Windows\PowerShell\Examples\Demo-ScriptArguments-ByNameParams.ps1

Script Block

More than one statement can be executed on one line by separating the statements with a semicolon. However those statements don't count as a script block.

```
PS> Set-Location variable:; Get-ChildItem

# Output
----                            -----
$                               }
?                               True
^                               {
args                            {}
arrayArguments                  {temp1.txt, temp2.txt}
ConfirmPreference               High
...
```

(Bentley, n.d.) 2015-07-20 13:06

A script block has the following syntax.

```
{
    [param ([type]$parameter1 [,[type]$parameter2])]
    <statement list>
}
```

```
PS> { Set-Location variable:; Get-ChildItem }
```

(Microsoft, 2015. Core Modules in Windows PowerShell) about_Script_Blocks, https://technet.microsoft.com/en-us/library/hh847893.aspx

Script blocks return either a single object or an array.

(Microsoft, 2015. Core Modules in Windows PowerShell) about_Script_Blocks, https://technet.microsoft.com/en-us/library/hh847893.aspx

For commands that take script blocks as a parameter, you can pass a script block.

```
C:\PS> invoke-command -scriptblock { param ($uu = "Parameter");
    "$uu assigned." }
Parameter assigned.
```

(Microsoft, 2015. Core Modules in Windows PowerShell) about_Script_Blocks, https://technet.microsoft.com/en-us/library/hh847893.aspx

See also: Script blocks and the Call Operator.

Script Language

Basics

Reference

The best Powershell script language syntax reference is: ...

This is especially so in the apparent absence of an official reference from Microsoft (without having to trawl through the get-help entries).

(Sheppard, 2015. Windows PowerShell command line syntax) http://ss64.com/ps/syntax.html

File Types

Scripts are text files saved with a *.ps1 extension.

(Microsoft, 2014. Scripting with Windows PowerShell) Running Scripts, https://technet.microsoft.com/en-us/library/bb613481%28v=vs.85%29.aspx

Comments

General

Comments. Inline "#". Block comments: "<#...#>".

```
# My great code.
Write-Host 'Starting Play01';

<#
  This is a block comment.

  Cool.
#>
```

C:\Users\John\Documents\Sda\Code\Windows\PowerShell\Examples\Demo-Reference.ps1

For comment based help, in general:

- Place comment based help in a block comment (although the system would also pick up on inline comments).
- .KEYWORD defines each section of comment-based help. There are a specific set of keywords you can use (e.g. .SYNOPSIS, .DESCRIPTION, etc.) The system automatically generates some sections when the user calls help (e.g. NAME, SYNTAX, etc.)

(Microsoft, 2015. Core Modules in Windows PowerShell) about_Comment_Based_Help, https://technet.microsoft.com/en-us/library/hh847834.aspx

Script

Comment-based help in scripts. Place at the beginning of the script file. Leave two lines between the end of the script block and any code (strictly only important if the code is a function).

```
<#

.SYNOPSIS
    A brief description of the function or script.

.DESCRIPTION
    A detailed description of the function or script.

.PARAMETER Parameter01
    The description of a parameter01.

    Windows PowerShell interprets all text between the .PARAMETER
    line and the next keyword or the end of the comment block as part of
    the parameter description. The description can include paragraph breaks.

    The Parameter keywords can appear in any order in the comment block, but
    the function or script syntax determines the order in which the parameters
    (and their descriptions) appear in help topic. To change the order,
    change the syntax.
```

```
.PARAMETER Parameter02
    The description of a parameter02.

.INPUTS
    None. You cannot pipe objects to this.

.OUTPUTS
    None. This does not generate any output.

.EXAMPLE
    PS> Get-Help .\Demo-CommentBasedHelp.ps1 -Full

    Calls the script to show the comments.

.EXAMPLE
    PS> .\Demo-CommentBasedHelp.ps1

    Calls the script.

#>


# Leave two lines above so the comment-based help is interpreted.
param ($parameter01, [String] $parameter02)
Write-Output 'Well, Hi';
```

(Microsoft, 2015. Core Modules in Windows PowerShell) about_Comment_Based_Help, https://technet.microsoft.com/en-us/library/hh847834.aspx
C:\Users\John\Documents\Sda\Code\Windows\PowerShell\Examples\Demo-CommentBasedHelp.ps1

Functions

Comment-based help in functions. Place before the function keyword or at the beginning of the function body.

```
<#
.SYNOPSIS
    My-Function01 comment-based help - Before the Function keyword.

.DESCRIPTION
    My-Function01 in lyrical terms.

.PARAMETER paramA
    Pass anything you fucking want.

.EXAMPLE
    PS> Get-Help My-Function01

    Demo the comment-based help.

.EXAMPLE
    PS> My-Function01

    Call the actual function.
#>
function My-Function01 ($paramA, [Int] $paramB) {
    "Nice My-Function01 $paramA"
}

function My-Function02 {
    <#
    .SYNOPSIS
        My-Function01 comment-based help - At the beginning of the function body.

    .DESCRIPTION
        My-Function02 in lyrical terms.
    #>

    'Cool My-Function02'
}
```

(Microsoft, 2015. Core Modules in Windows PowerShell) about_Comment_Based_Help, https://technet.microsoft.com/en-us/library/hh847834.aspx
C:\Users\John\Documents\Sda\Code\Windows\PowerShell\Examples\Demo-CommentBasedHelp.ps1

Case

Script is case insensitive. However, you can use tab completion features of the command line or the ISE to Capitalize-TheCommand.

```
# Script is case insensitive
Write-Host 'Capital command works'
write-host 'lower case command works'
```

C:\Users\John\Documents\Sda\Code\Windows\PowerShell\Examples\Demo-Reference.ps1

Statement seperator

Statements are separated (not terminated). Carriage return/newline or (optionally) semicolon.

```
# Statement separater: semicolon
Write-Host 'Starting Play01'; Write-Host 'Starting Play02';

# Statement separater: carriage return/newline
write-host 'Second line'
write-host 'third line'
```

C:\Users\John\Documents\Sda\Code\Windows\PowerShell\Examples\Demo-Reference.ps1
(Microsoft, 2015. Core Modules in Windows PowerShell) about_Functions, https://technet.microsoft.com/en-us/library/hh847829.aspx

Line continuation character

In scripts the line continuation character is the backtick "`".

```
Copy-Item -Exclude *.html, *.xhtml, *.xml, *.bat, *.ps1 -Recurse `
| Where {$_.FullName -notlike "*\xwmEngine*"} `
| format-table -AutoSize -Property mode, FullName
```

Escape character

The escape character is used either in strings or in statements to prevent the normal interpretation of the subsequent character. The escape characters is the backtick (`) or, to prevent powershell from interpreting arguments use the stop-parsing symbol (--%).

```
``  To avoid using a Grave-accent as the escape character
`#  To avoid using # to create a comment
`'  To avoid using ' to delimit a string
`"  To avoid using " to delimit a string
```

See Parameters, supplying for stop-parsing examples.
(Microsoft, 2015. Core Modules in Windows PowerShell) about_Escape_Characters, https://technet.microsoft.com/en-us/library/hh847755.aspx
(Sheppard, 2015. Windows PowerShell command line syntax) Escape characters, Delimiters and Quotes, http://ss64.com/ps/syntax-esc.html

Include or Import other scripts

To include other scripts ensure you precede the referenced script with `Set-Location $PSScriptRoot`, and use the Dot Sourcing operator (.[space]) to ensure the StandardApplicationLibrary.ps1 functions are retained in the current context.

```
Set-Location $PSScriptRoot

# dot sourcing operator ensures functionality in the called script persists
# for the session.
. ..\Libraries\StandardApplicationLibrary.ps1

# or
. C:\Users\John\Documents\Sda\Code\PowerShell\Libraries\StandardApplicationLibrary.ps1

function Demo-TryCatchFinally {
...
```

(Bentley, n.d.) 2015-08-03 11:33

Functions

Powershell has built in functions (e.g. Out-Speech, Edit-File, prompt, etc.) and you can write your own.

See Function discovery and help.

You write functions in these places:

- At the command line. In that case you can only resuse your function for the duration of the session.
- Put it in your profile files, or
- In script (which you could call with the dot source operator to make the function available to the session).

(Microsoft, 2015. Core Modules in Windows PowerShell) Using Functions, https://technet.microsoft.com/en-us/library/dd745030%28v=vs.85%29.aspx

Function syntax.

```
function [<scope:>]<name> [([type]$parameter1[,[type]$parameter2])]
{
    # If you don't declare parameters above the opening bracket.
    param([type]$parameter1 [,[type]$parameter2])

    dynamicparam {<statement list>}

    begin {<statement list>}
    process {<statement list>}
    end {<statement list>}
}
```

(Microsoft, 2015. Core Modules in Windows PowerShell) about_Functions, https://technet.microsoft.com/en-us/library/hh847829.aspx

Functions should be named according the Verb-Noun convention; using the convention for verbs. See: (Microsoft, 2015. Approved Verbs for Windows PowerShell Commands) https://msdn.microsoft.com/en-us/library/ms714428%28VS.85%29.aspx

(Microsoft, 2015. Core Modules in Windows PowerShell) about_Functions, https://technet.microsoft.com/en-us/library/hh847829.aspx

If you want to process objects in a pipeline then you need to declare a process {} clause.

```
# Do this
function Write-Pipeline { process {"The value is: $_"} }
1, 2, 4 | Write-Pipeline

# Output
The value is: 1
The value is: 2
The value is: 4

# Not this
function Write-Pipeline { "The value is: $_" }
1, 2, 4 | Write-Pipeline

# Output (pipline not processed)
The value is:
```

(Microsoft, 2015. Core Modules in Windows PowerShell) about_Functions, https://technet.microsoft.com/en-us/library/hh847829.aspx
(Bentley, n.d.) 2015-07-20 09:20

Filters

Alternatively, declare a filter, which is just a function that has an implicit process block (and no begin and end blocks).

```
filter Write-Pipeline { "The value is: $_"}
1, 2, 4 | Write-Pipeline

# Output
The value is: 1
The value is: 2
The value is: 4
```

(Microsoft, 2015. Core Modules in Windows PowerShell) about_Functions, https://technet.microsoft.com/en-us/library/hh847829.aspx
(Bentley, n.d.) 2015-07-20 09:23

Function Parameters

General

Function parameters can be: positional, named, switch, or dynamic.

(Microsoft, 2015. Core Modules in Windows PowerShell) about_Functions, https://technet.microsoft.com/en-us/library/hh847829.aspx

By convention declare the parameters in PascalCase.

```
function Get-CoolThing ([int] $Size) {
    Get-ChildItem | Where {$_.Length -lt $Size -and !$_.PSIsContainer}
}

Get-CoolThing -Size 10000
```

Set a default parameter value with "= x".

```
function Get-CoolThing ($Size = 100) {
    Get-ChildItem | Where {$_.Length -lt $Size -and !$_.PSIsContainer}
}
```

(Microsoft, 2015. Core Modules in Windows PowerShell) about_Functions, https://technet.microsoft.com/en-us/library/hh847829.aspx

To add help information about the default value use the PSDefaultValue attribute AND ensure you have some comment-base help.

```
# .SYNOPSIS
#    Does something cool
function Get-CoolThing ( [PSDefaultValue(Help = '200')] $Size = 200 )  {
    Get-ChildItem | Where {$_.Length -lt $Size -and !$_.PSIsContainer}
}

# .SYNOPSIS
#    Does something cool
function Get-CoolThing  {
    param ([PSDefaultValue(Help = '100')]$Size = 100)
    Get-ChildItem | Where {$_.Length -lt $Size -and !$_.PSIsContainer}
}
```

(Microsoft, 2015. Core Modules in Windows PowerShell) about_Functions, https://technet.microsoft.com/en-us/library/hh847829.aspx
(Bentley, n.d.) 2015-07-13 05:00

Positional arguments are arguments not named in the function. You pass any number of arguments to the function at call time. You access each argument within the function with `$args[0], $args[1], ... $args[n-1].`

```
function Demo-PositionalParameters {
    "The arguments: " +  $args[0] + " " + $args[1] + "."
}

Demo-PositionalParameters cool apples

# Result.
The arguments: cool apples.
```

(Microsoft, 2015. Core Modules in Windows PowerShell) about_Functions, https://technet.microsoft.com/en-us/library/hh847829.aspx
(Bentley, n.d.) 2015-07-13 15:13

If the function names parameters and accesses positional arguments then the named parameters take the initial position if not named when called.

```
function Demo-PositionalParameters ($Fun) {
    "The arguments: " +  $args[0] + " " + $args[1] + "."
}

Demo-PositionalParameters cool apples cats

The arguments: apples cats. # $Fun has consumed "cool".
```

(Microsoft, 2015. Core Modules in Windows PowerShell) about_Functions, https://technet.microsoft.com/en-us/library/hh847829.aspx
(Bentley, n.d.) 2015-07-13 15:13

Declare a switch parameter just by specifying type. Call by just passing the name of the parameter: that switches it "on (true)". If you don't pass the name of the parameter the value will be "off (false)".

```
# Declare
function Demo-SwitchParameter([switch]$IsCool) {
    if ($IsCool) {
        "You are cool."
    } else {
        "You are not cool."
    }
}

Demo-SwitchParameter -IsCool
```

```
# Returns "You are cool."

Demo-SwitchParameter
# Returns "You are not cool."
```

(Microsoft, 2015. Core Modules in Windows PowerShell) about_Functions, https://technet.microsoft.com/en-us/library/hh847829.aspx
(Bentley, n.d.) 2015-07-13 17:08

You can use splattings with the specials $args variable to pass paramaters along to another function.

```
function Jlb-Get-ChildItem { Get-ChildItem @args }
Jlb-Get-ChildItem F*, P*
```

(Microsoft, 2015. Core Modules in Windows PowerShell) about_Functions, https://technet.microsoft.com/en-us/library/hh847829.aspx
(Bentley, n.d.) 2015-07-20 09:14

Attributes

Function parameters can have attributes specified against them, including validation attributes.

```
function MM-Copy-Items {
    Param (
      [parameter(Mandatory=$true)]
      [String]
      $SourceDirectory,

      [parameter(Mandatory=$true)]
      [String]
      $DestinationDirectory,

      [parameter(Mandatory=$true)]
      [String[]]
      $Files
    )
    # Ensure directory exists, otherwise create it.
    New-Item $DestinationDirectory -ItemType Directory -Force
    Copy-Item -Path $SourceDirectory\* -Destination $DestinationDirectory -Include $Files
}

MM-Copy-Items -SourceDirectory $sourceDir -DestinationDirectory $tempDestDir -Files $files
```

C:\Users\John\Documents\Sda\Code\web\Apps\Softmake\Source\makeMaster.ps1

Common attributes.

```
# Parameter attribute with mandatory argument.
param
(
  [parameter(Mandatory=$true)]
  [String[]]
  $ComputerName
)

# Alias attributue
param
(
  [parameter(Mandatory=$true)]
  [alias("CN","MachineName")]
  [String[]]
  $ComputerName
)
```

```
# Validation attribute, AllowNull()
param
(
  [parameter(Mandatory=$true)]
  [AllowNull()]
  [String]
  $ComputerName
)

# Validation attribute, ValidateRange
param
(
  [parameter(Mandatory=$true)]
  [ValidateRange(0,10)]
  [Int]
  $Attempts
)
```

(Microsoft, 2015. Core Modules in Windows PowerShell) about_Functions_Advanced_Parameters, https://technet.microsoft.com/en-us/library/dd347600.aspx

Parameter attributes.

Category	**Attribute**
Parameter Attribute arguments	parameter(Mandatory=$true)
	parameter(Position=0)
	[parameter(Mandatory=$true, ParameterSetName="Computer")]
	[parameter(Mandatory=$true, **ValueFromPipeline**=$true)]
	[parameter(Mandatory=$true, ValueFromPipelineByPropertyName=$true)]
	[parameter(Mandatory=$true, **ValueFromRemainingArguments**=$true)]
	[parameter(mandatory=$true, HelpMessage="Enter one or more computer names separated by commas.")]
Alias Attribute	[alias("CN","MachineName")]
Parameter Validation Attributes	[AllowNull()]
	[AllowEmptyString()]
	[AllowEmptyCollection()]
	[ValidateCount(1,5)]
	[ValidateLength(1,10)]
	[ValidatePattern("[0-9][0-9][0-9][0-9]")]
	[ValidateRange(0,10)]
	[ValidateScript({$_ -ge (get-date)})]
	[ValidateSet("Low", "Average", "High")]
	[ValidateNotNull()]
	[ValidateNotNullOrEmpty()]

(Microsoft, 2015. Core Modules in Windows PowerShell) about_Functions_Advanced_Parameters, https://technet.microsoft.com/en-us/library/dd347600.aspx

Pass arguments by Reference

Pass arguments by referencing using [ref] both at the passing argument and the receiving parameter. Use the Value property of the System.Management.Automation.PSReference object for assignment. Declare the variable before passing it by reference.

```
$message = "love"

function Transform-Message {
    param (
        [parameter(Mandatory=$true)]
        [ref]
        $MessageToTransform
    )

    # System.Management.Automation.PSReference must be assigned
    # through Value property
    $MessageToTransform.Value = $MessageToTransform.Value + " all"
}

# $message is declared before being passed.
$message
Transform-Message ([ref]$message)
# Or Transform-Message -MessageToTransform ([ref]$message)
$message

# Output
PS> .\Demo-PassByReference.ps1
love
love all
```

(Bentley, n.d.) 2015-09-15 20:32
"C:\Users\John\Documents\Sda\Code\Windows\PowerShell\Examples\Demo-PassByReference.ps1"

Control Flow Statements

If

If syntax.

```
if (<test1>)
    {<statement list 1>}
[elseif (<test2>)
    {<statement list 2>}]
[else
    {<statement list 3>}]
```

```
if ($a -gt 2) {
    Write-Host "The value $a is greater than 2."
} elseif ($a -eq 2) {
    Write-Host "The value $a is equal to 2."
} else {
    Write-Host "The value $a is less than 2 or was not created or initialized."
}
```

(Microsoft, 2015. Core Modules in Windows PowerShell) about_If, https://technet.microsoft.com/en-us/library/hh847876.aspx

Switch

Switch Syntax.

```
    switch [-regex|-wildcard|-exact][-casesensitive] (<value>)

or

    switch [-regex|-wildcard|-exact][-casesensitive] -file filename

followed by

    {
        "string"|number|variable|{ expression } { statementlist }
        default { statementlist }
    }
```

(Microsoft, 2015. Core Modules in Windows PowerShell) about_Switch, https://technet.microsoft.com/en-us/library/hh847750.aspx

> If no parameters are used, Switch performs a case-insensitive exact match for the value. If the value is a collection, each element is evaluated in the order in which it appears.

By default all the test expressions are evaluated any all matches are returned.

```
PS> switch (3)
{
  1 {"It is one."}
  2 {"It is two."}
  3 {"It is three."}
  4 {"It is four."}
  3 {"Three again."}
}
It is three.
Three again.
```

(Microsoft, 2015. Core Modules in Windows PowerShell) about_Switch, https://technet.microsoft.com/en-us/library/hh847750.aspx

Use "Break" to halt further evaluation of test expressions, when a condition is matching.

```
PS> switch (3)
 {
    1 {"It is one."}
    2 {"It is two."}
    3 {"It is three."; Break}
    4 {"It is four."}
    3 {"Three again."}
 }
It is three.
```

(Microsoft, 2015. Core Modules in Windows PowerShell) about_Switch, https://technet.microsoft.com/en-us/library/hh847750.aspx

Switches can handle collections. Collections are processed in element order. The whole switch statement is terminated by a break.

```
# Evaluate each collection element in order.
PS> switch (4, 2)
 {
    1 {"It is one." }
    2 {"It is two." }
    3 {"It is three." }
    4 {"It is four." }
    3 {"Three again."}
 }
It is four.
It is two.

# Halt exectution of entire switch when you reach a 'Break'.
PS> switch (4, 2)
{
```

```
  1 {"It is one."; Break}
  2 {"It is two." ; Break }
  3 {"It is three." ; Break }
  4 {"It is four." ; Break }
  3 {"Three again."}
}
It is four
```

(Microsoft, 2015. Core Modules in Windows PowerShell) about_Switch, https://technet.microsoft.com/en-us/library/hh847750.aspx

Use a switch with strings.

```
$target = 'https://bing.com'
switch -Regex ($target)
{
    '^ftp\://.*$' { "$_ is an ftp address"; Break }
    '^\w+@\w+\.com|edu|org$' { "$_ is an email address"; Break }
    '^(http[s]?)\://.*$' { "$_ is a web address that uses $($matches[1])"; Break }
}

https://bing.com is a web address that uses https
```

(Microsoft, 2015. Core Modules in Windows PowerShell) about_Switch, https://technet.microsoft.com/en-us/library/hh847750.aspx

Default clause.

```
switch ("fourteen")
{
    1 {"It is one."; Break}
    2 {"It is two."; Break}
    3 {"It is three."; Break}
    4 {"It is four."; Break}
    "fo*" {"That's too many."}
    Default {
        "No matches"
    }
}

No matches
```

(Microsoft, 2015. Core Modules in Windows PowerShell) about_Switch, https://technet.microsoft.com/en-us/library/hh847750.aspx

While

While Syntax.

```
while (<condition>){<statement list>}
```

```
$val = 0
while ($val -ne 3) {
    $val++
    Write-Host $val
}
```

(Microsoft, 2015. Core Modules in Windows PowerShell) about_While, https://technet.microsoft.com/en-us/library/hh847745.aspx

Do

Do loops execute at least once. Do-Whiles continue iteration while the condition is true. Do-Untils continue iteration while the condition is false (until the condition is true).

(Microsoft, 2015. Core Modules in Windows PowerShell) about_Do, https://technet.microsoft.com/en-us/library/hh847887.aspx

Do Syntax.

```
do {<statement list>} while (<condition>)

do {<statement list>} until (<condition>)
```

```
$x = 0
do {
    $x++
    Write-Host $x
} while ($x -lt 3)

$x = 0
do {
    $x++
    Write-Host $x
} until ($x -eq 3)
```

(Microsoft, 2015. Core Modules in Windows PowerShell) about_Do, https://technet.microsoft.com/en-us/library/hh847887.aspx

For

For syntax.

```
for (<init>; <condition>; <repeat>)
{<statement list>}

<init> and <repeat> can have several statements, separated by a comma. However, comma
separation doesn't work in the <init> clause for assignments.

Alternative syntax uses a carriage return, rather than a semi-colon (;), to separate <init>,
<condition>, and <repeat>:

for (<init>
<condition>
<repeat>){
  <statement list>
}
```

```
for ($i=1; $i -le 5; $i++) {
    $i
}
```

(Microsoft, 2015. Core Modules in Windows PowerShell) about_For, https://technet.microsoft.com/en-us/library/hh847870.aspx

Foreach

Foreach has a script syntax and a different pipleline syntax.

See ForEach

Currently Running Script Name

To get the currently running script name, whether you call directly in the script or within a function, use `$PSCommandPath`.

```
## Test.ps1
'Direct:'
$PSCommandPath  # Full Path
```

```
Split-Path -Path $PSCommandPath -Leaf  # File Name only

function main () {
  ''
  'Within a function:'
  $PSCommandPath
  Split-Path -Path $PSCommandPath -Leaf
}

main

## Result
PS> .\Test.ps1
Direct:
C:\Users\John\Documents\Sda\Code\Windows\PowerShell\Apps\xBankStatementRename\Test.ps1
Test.ps1

Within a function:
C:\Users\John\Documents\Sda\Code\Windows\PowerShell\Apps\xBankStatementRename\Test.ps1
Test.ps1
```

StackOverflow, "How can I get the current PowerShell executing file?", gregmac's answer:
https://stackoverflow.com/a/43643346/872154
See also my (John Bentley's) answer: https://stackoverflow.com/a/65846536/872154

C:\Users\John\Documents\Sda\Code\Windows\PowerShell\Examples\Demo-PathAndFileNameParts.ps1s

Error Handling

Standard Practice

Try-Catch-Finally

Use the following try, catch, finally pattern with Sal-Out-LastError from \Libraries\StandardApplicationLibrary.ps1 (John Bentley Created). By default Sal-Out-LastError will output an error message that enables you, as developer, to readily identify the error causing exception and any *inner* exception.

```
Set-Location $PSScriptRoot
. ..\Libraries\StandardApplicationLibrary.ps1

function Demo-TryCatchFinally {
    try {
        $wc = new-object System.Net.WebClient
        $wc.DownloadFile('http://www.example.com/MyDoc.doc', 'MyDoc.doc')

    } catch [System.Net.WebException],[System.IO.IOException] {
        Sal-Out-LastError -UserMessage 'Unable to download MyDoc.doc from
http://www.example.com.'

    } catch { # Default
        Sal-Out-LastError -UserMessage 'An error occurred that the developers did not
anticipate.'

    } finally {
      'End of processing'
    }
}
...

PS C:\Users\John\Documents\Sda\Code\Windows\PowerShell\Examples> .\Demo-ErrorHandling.ps1
********** An error occurred. **********

For the user ...
Unable to download MyDoc.doc from http://www.example.com.

For the developer ...
Exception calling "DownloadFile" with "2" argument(s): "The remote server returned an error:
(404) Not Found."
```

```
$LastError.InvocationInfo.PositionMessage ...
At C:\Users\John\Documents\Sda\Code\Windows\PowerShell\Examples\Demo-ErrorHandling.ps1:7
char:9
+         $wc.DownloadFile('http://www.example.com/MyDoc.doc', 'MyDoc.doc')
+         ~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~

$LastError.CategoryInfo:                                NotSpecified: (:) [],
MethodInvocationException
$LastError.FullyQualifiedErrorId:                       WebException

$LastError.Exception.GetType().FullName:
System.Management.Automation.MethodInvocationException
$LastError.Exception.Message:                           Exception calling "DownloadFile"
with "2" argument(s): "The remote server returned an error: (404) Not Found."

$LastError.Exception.InnerException.GetType().FullName: System.Net.WebException
$LastError.Exception.InnerException.Message:            The remote server returned an error:
(404) Not Found.
**************************************

End of processing
```

(Bentley, n.d.) *\\ATLAS\Documents\Sda\Code\Windows\PowerShell\Examples\Demo-ErrorHandling.ps1*
(Bentley, n.d.) *\\ATLAS\Documents\Sda\Code\Windows\PowerShell\Libraries\StandardApplicationLibrary.ps1*

Throwing Exceptions

The string specified in a throw is for the developer rather than the developer, even though the official documentation refers to this string as a "user message". You ought generally use a try-catch-finally clause with Sal-Out-LastError (from John Bentley's StandardApplicationLibrary.ps1) to wrap the Exception and supply a user message.

```
Set-Location $PSScriptRoot
. ..\Libraries\StandardApplicationLibrary.ps1

function Demo-Throw {
    try {
        throw New-Object System.FormatException "XML was not well formed."

    } catch [System.FormatException]{
        Sal-Out-LastError -UserMessage `
            'The document you received as not in the correct format.'

    } catch { # Default
        Sal-Out-LastError -UserMessage `
            'An error occurred that the developers did not anticipate.'

    } finally {
        'End of processing'
    }
}

Demo-Throw

PS> .\Demo-ErrorHandling.ps1
********** An error occurred. **********

For the user ...
The document you received as not in the correct format.

For the developer ...
XML was not well formed.

$LastError.InvocationInfo.PositionMessage ...
At C:\Users\John\Documents\Sda\Code\Windows\PowerShell\Examples\Demo-ErrorHandling.ps1:54
char:9
+         throw New-Object System.FormatException "XML was not well formed."
+         ~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~
```

```
$LastError.CategoryInfo:                                OperationStopped: (:) [],
FormatException
$LastError.FullyQualifiedErrorId:                       XML was not well formed.

$LastError.Exception.GetType().FullName:                System.FormatException
$LastError.Exception.Message:                           XML was not well formed.
****************************************

End of processing
```

(Bentley, n.d.) 2015-08-04 12:35

Catching non-terminating errors

There are two methods for enabling non-terminating errors to be caught by a try, catch, finally block:

1. Bookend candidate statements with $ErrorActionPreference = 'Stop' and $ErrorActionPreference = 'Continue';

```
try {
    $ErrorActionPreference = "Stop"
    Get-ChildItem Fun.txt
} catch [System.Management.Automation.ItemNotFoundException] {
    Sal-Out-LastError -UserMessage 'The file does not exist.'

} catch { # Default
    Sal-Out-LastError -UserMessage 'An error occurred that the developers did not
anticipate.'

} finally {
    $ErrorActionPreference = "Continue"
    'finally done.'
}
```

2. Use the common parameter and value: `-ErrorAction Stop`

```
try {

    Get-ChildItem Fun.txt -ErrorAction Stop

} catch [System.Management.Automation.ItemNotFoundException] {
    Sal-Out-LastError -UserMessage 'The file does not exist.'

} catch { # Default
    Sal-Out-LastError -UserMessage 'An error occurred that the developers did not
anticipate.'

} finally {
    'finally done.'
}
```

(Bentley, n.d.) 2015-08-03 22:27
(Babinec, 2013. An Introduction to Error Handling in PowerShell) http://blogs.msdn.com/b/kebab/archive/2013/06/09/an-introduction-to-error-handling-in-powershell.aspx

Overview

A `try, catch, finally` block can be used to handle errors.

```
try {
    $wc = new-object System.Net.WebClient
    $wc.DownloadFile('http://www.example.com/MyDoc.doc', 'MyDoc.doc')

} catch [System.Net.WebException],[System.IO.IOException] {
    'Unable to download MyDoc.doc from http://www.example.com.'
```

```
} catch { # Default
    'An error occurred that the developers did not anticipate.'
    $Error[0]

} finally {
    'End of processing'
}
```

(Babinec, 2013. An Introduction to Error Handling in PowerShell) http://blogs.msdn.com/b/kebab/archive/2013/06/09/an-introduction-to-error-handling-in-powershell.aspx
(Microsoft, 2015. Core Modules in Windows PowerShell) about_Try_Catch_Finally, https://technet.microsoft.com/en-us/library/hh847793.aspx
(Bentley, n.d.) 2015-08-03 11:39

You can use try-catch-finally at the script level, as well as the function level.

```
#xCommon_Profifle.ps1
try{
    Set-Alias -Name gh -Value Get-Help
} catch {
    throw $Error[0]
} finally {
    "xCommon_Profile.ps1 finished OK."
}
```

(Bentley, n.d.) \\ATLAS\Documents\WindowsPowerShell\xCommon_profile.ps1

There are two types of errors, terminating and non-terminating.

(Microsoft, 2014. Scripting with Windows PowerShell) Managing Errors, https://technet.microsoft.com/en-us/library/bb648602%28v=vs.85%29.aspx
(Babinec, 2013. An Introduction to Error Handling in PowerShell) http://blogs.msdn.com/b/kebab/archive/2013/06/09/an-introduction-to-error-handling-in-powershell.aspx

By default only terminating errors can be caught by a try, catch, finally block.

(Babinec, 2013. An Introduction to Error Handling in PowerShell) http://blogs.msdn.com/b/kebab/archive/2013/06/09/an-introduction-to-error-handling-in-powershell.aspx

Examining the last error

Powershell maintains the variable $Error, an array of all the errors generated in the current session. The later errors appear earlier in the array. Therefore $Error[0] contains the last error generated. Call $Error[0] to have powershell display the last error in a default format.

(Babinec, 2013. An Introduction to Error Handling in PowerShell) http://blogs.msdn.com/b/kebab/archive/2013/06/09/an-introduction-to-error-handling-in-powershell.aspx

Handy last error interrogative commands.

```
$Error[0].Exception.GetType().FullName
$Error[0].Exception.InnerException.GetType().FullName
$Error[0] | Format-List -Force
$Error[0].Exception | Format-List -Force
```

(Wories, 2009. PowerShell Tips & Tricks: Getting more detailed error information from PowerShell)
(Bentley, n.d.) 2015-08-03 08:39

Throw

Throw syntax.

```
throw [<expression>]
```

(Microsoft, 2015. Core Modules in Windows PowerShell) about_Throw, https://technet.microsoft.com/en-us/library/hh847766.aspx

Throw generates different types of errors depending on the supplied expression.

Syntax	Example	Result
throw <no expression>	throw	ErrorRecord > Exception = RuntimeException > Exception.Message = "ScriptHalted"
throw <string>	throw "Error message"	ErrorRecord > Exception = RuntimeException > Exception.Message = "Error message"
throw <.Net Framework Exception>	throw New-Object ` System.FormatException	ErrorRecord > Exception = <Specified .Net Exception> > Exception.Message = "Native message for specified .Net Exception."
throw <.Net Framework Exception> <string>	throw New-Object ` System.FormatException ` "Error message"	ErrorRecord > Exception = <Specified .Net Exception> > Exception.Message = "Error Message."

(Microsoft, 2015. Core Modules in Windows PowerShell) about_Throw, https://technet.microsoft.com/en-us/library/hh847766.aspx
(Bentley, n.d.) 2015-08-04 12:23

Handling errors from External Programs

To handle errors from external programs you interrogate the $LASTEXITCODE. It's then a matter of reading the external program documentation to discover the meaning of the exit code.

```
function Demo-HandleExternalProgram {
    # robocopy works on folders, not files as such.
    # This will exit an write to $LastErrorCode
    robocopy ".\temp1.txt" ".\temp3.txt"
    if ($LASTEXITCODE -ne 0) {
        Write-Host $('$LASTEXITCODE {0} indicates an error' -f $LASTEXITCODE) -
ForegroundColor Red
    }
}

PS> .\Demo-ErrorHandling.ps1

------------------------------------------------------------------------------
   ROBOCOPY     ::     Robust File Copy for Windows
------------------------------------------------------------------------------

  Started : Monday, 3 August 2015 22:51:30
   Source : C:\Users\John\Documents\Sda\Code\Windows\PowerShell\Examples\temp1.txt\
     Dest : C:\Users\John\Documents\Sda\Code\Windows\PowerShell\Examples\temp3.txt\

    Files : *.*

  Options : *.* /DCOPY:DA /COPY:DAT /R:1000000 /W:30
```

```
------------------------------------------------------------------------------

2015/08/03 22:51:30 ERROR 123 (0x0000007B) Accessing Source Directory
C:\Users\John\Documents\Sda\Code\Windows\PowerShell\Examples\temp1
.txt\
The filename, directory name or volume label syntax is incorrect.

$LASTEXITCODE 16 indicates an error.
```

(Bentley, n.d.) 2015-08-03 23:00
(Babinec, 2013. An Introduction to Error Handling in PowerShell) http://blogs.msdn.com/b/kebab/archive/2013/06/09/an-introduction-to-error-handling-in-powershell.aspx

When error handling an external command you can supress it's output by piping to `Out-Null`, although generally you wouldn't want to.

```
function Demo-HandleExternalProgram {
    # robocopy works on folders, not files as such.
    # This will exit an write to $LastErrorCode
    # Out-Null supresses output from external command.
    robocopy ".\temp1.txt" ".\temp3.txt" | Out-Null
    if ($LASTEXITCODE -ne 0) {
        Write-Host $('$LASTEXITCODE {0} indicates an error' -f $LASTEXITCODE) -
ForegroundColor Red
    }
}

Demo-HandleExternalProgram

PS> .\Demo-ErrorHandling.ps1
$LASTEXITCODE 16 indicates an error
```

(Bentley, n.d.) 2015-08-03 23:02
Stackoverflow, Suppress output from non-PowerShell commands?, Answer Ansgar Wiechers 2013-05-24 22:52, http://stackoverflow.com/a/16744656

Sometimes an array of error records are created by an external command.

```
$openSslErrorRecord = openssl.exe ca -batch `
    -config $mOpenSslConfigurationFile `
    -in $mUserCertificateFileCertificateSigningRequest `
    -out $mUserCertificateFileCertificate `
    -outdir ($mUserCertificateDirectory + 'out\')  2>&1

$openSslErrorRecord | Format-List -Force

# Can display the properties for three error record objects.
```

(Bentley, n.d.) 2015-08-23 14:50

Sometimes you have to force an error to write to the "success or error stream" in order for the error to be catchable from both the Powershell ISE and Native Powershell. Do this with "2>&1".

```
function Jlb-New-UserCertificateCASigned () {

        try {
            openssl.exe ca -batch `
                -config $mOpenSslConfigurationFile `
                -in $mUserCertificateFileCertificateSigningRequest `
                -out $mUserCertificateFileCertificate `
                -outdir ($mUserCertificateDirectory + 'out\') 2>&1


        } catch { # default
            Write-Host "Error Occurred and Caught" -ForegroundColor Red
            Write-Output $Error[0]
            Exit
```

```
        } finally {
            $ErrorActionPreference = "Continue"
        }

        $script:mUserCertificateCreatedFlag = $true
    }
    ''
}
```

(Bentley, n.d.) 2015-08-25 16:01

Create an Exit Code

Create an Exit Code as follows. By convention Exit 0 means "OK".

```
function Demo-ExitCode {
    "Demoing ..."
    Exit 34
}

Demo-ExitCode

PS> .\Demo-ErrorHandling.ps1
Demoing ...

PS> $LASTEXITCODE
34
```

(Bentley, n.d.) \\ATLAS\Documents\Sda\Code\Windows\PowerShell\Examples\Demo-ErrorHandling.ps1
http://blogs.msdn.com, PowerShell Team, Windows PowerShell Exit Codes,
http://blogs.msdn.com/b/powershell/archive/2006/10/14/windows-powershell-exit-codes.aspx

Working in Powershell

Navigation

Useful Navigation commands.

Command(s)	Notes
Push-Location, Pop-Location	Save and recall locations
Set-Location, Get-Location	
Get-PSDrive, New-PSDrive, Remove-PSDrive	

(Microsoft, 2014. Scripting with Windows PowerShell) Managing Windows PowerShell Drives, https://technet.microsoft.com/en-us/library/dd315335.aspx

Keyboard Shortcuts

Common

Command	Key
Code complete	Tab{1, 3}
Clear the current line	Esc
Continue to next line without exectution	Shift + Enter

ISE

Command	Key
Console Window: Page up or Page down.	Ctrl + [Page Up | Page Down]
Toggle Script Window	Ctrl + R

Techniques

Techniques Reference

For standard techniques that involve managing:

- Powershell Current Location.
- Powershell Drives.
- Powershell Item Manipulation.
- File system.
- Processes.
- Services.
- Computer and System Info.
- Software installations.
- Printers.
- Networking.
- Registry.
- Remote Commands.

See ...

(Microsoft, 2014. Scripting with Windows PowerShell) Windows PowerShell Navigation, https://technet.microsoft.com/en-us/library/dd315257.aspx
(Microsoft, 2014. Scripting with Windows PowerShell) Using Windows PowerShell for Administration, https://technet.microsoft.com/en-us/library/dd315367.aspx

Prompt

Change the prompt.

```
function prompt { 'PS> ' }
function prompt { 'PS C:\Users\John\Documents\Git\Examples\Spoon-Knife> ' }
function prompt { 'PS ' + $(get-location) + '> ' }

// The last two folders in the path
function prompt { 'PS ...' + (( $(get-location) -split '\\' | select -last 2) -join '\') +
'> '}
// E.g.
PS ..Examples\basic-demo>
```

(Sheppard, 2015. Windows PowerShell command line syntax) PowerShell Prompt, http://ss64.com/ps/syntax-prompt.html

Curl

For curl usage, use Invoke-WebRequest (iwr).

```
PS> Invoke-WebRequest -Uri http://localhost:8080/

StatusCode        : 200
StatusDescription : OK
Content           : <!DOCTYPE html PUBLIC "-//W3C//DTD XHTML 1.1//EN"
                        "http://www.w3.org/TR/xhtml11/DTD/xhtml11.dtd">
                    <!--
                      Omit XML declaration from top line as it puts IE6 in quirks mode.
                      Since the XML declar...
RawContent        : HTTP/1.1 200 OK
[etc..]


PS> Invoke-WebRequest -Uri http://localhost:8080/ | Format-List -Property Content

Content : <!DOCTYPE html PUBLIC "-//W3C//DTD XHTML 1.1//EN"
```

```
            "http://www.w3.org/TR/xhtml11/DTD/xhtml11.dtd"> [etc..]


PS> Invoke-WebRequest -Uri http://localhost:8080/ | Select-Object -ExpandProperty Content

<!DOCTYPE html PUBLIC "-//W3C//DTD XHTML 1.1//EN"
  "http://www.w3.org/TR/xhtml11/DTD/xhtml11.dtd"> [etc..]
```

(Bentley, n.d.) 2016-05-18 15:22
See http://superuser.com/a/994132/226936

Download a file from the internet. Use Powershell 6 which uses a default encoding of utf8NoBOM

```
# Simple
> Invoke-WebRequest -Uri
https://raw.githubusercontent.com/fancyapps/fancybox/master/dist/jquery.fancybox.min.js -
OutFile jquery.fancybox.min.js

# More control
> Invoke-WebRequest -Uri
https://raw.githubusercontent.com/fancyapps/fancybox/master/dist/jquery.fancybox.min.js |
Select-Object -ExpandProperty Content | Out-File jquery.fancybox.min.js -Encoding utf8NoBOM
```

Only from Powershell 6.0 does the encoding default to UTF8 with No BOM.

Out-File https://docs.microsoft.com/en-us/powershell/module/microsoft.powershell.utility/out-file?view=powershell-6#parameters

Powershell version

Get the powershell version.

```
$PSVersionTable
```

(Bentley, n.d.) 2019-09-16

Get the powershell installation paths.

```

```

File System Operations

Get all child files and directories recursively, but exclude a directory and its contents.

```
Get-ChildItem .\ -Recurse `
| Where {$_.FullName -notlike "*\xwmEngine*"} `
| format-table -AutoSize -Property mode, FullName -GroupBy Directory
```

Copy all child files and directories recursively, but exclude a directory include only some types of files. Use robocopy, powershell copy-item -recurse is fucked.

```
robocopy . $destDir *.ico, *.css, *.pdf, *.js *.htaccess, *.txt, *.gz /xd xwmEngine /s /np
```

Create a directory if it doesn't exist:

- C:\Users\John\Documents\Sda\Code\Windows\PowerShell\Libraries\StandardApplicationLibrary.ps1, Sal-New-Directory; or
- The following ....

```
$path = ".\output"
```

```
If(!(Test-Path $path))
{
    New-Item -ItemType Directory -Force -Path $path | Out-Null
    $path  + " created."
}
```

C:\Users\John\Documents\Sda\Code\Windows\PowerShell\Examples\Demo-FileSystemOperations.ps1

PackageManagement

Overview

PackageManagement, formerly "OneGet", is Microsoft's Package Management Manager. It exposes a consistent framework to work with many package managers. A package manager entails an install technology and, often, a centralized repository of packages on the web. A package manager facilitates local and remote software discovery, installation, inventory and update. "PackageManagement" is implemented as a PowerShell (5.0) module.

PackageManagement parts.

Part	Description	Example	CmdLets
Provider	Package Managers are exposed as "providers". A provider is an install technology. Each provider may manage one or more package sources.	Programs msu msi PSModule NuGet Appx Python PHP	Get-PackageProvider
Source	If local, a directory. If remote, a url to a repository. Sources are always managed by a provider.	PowerShellGallery NuGet Gallery 1 NUGET Galler 2	Get-PackageSource Register-PackageSource Set-PackageSource Unregister-PackageSource
Package	A local or remote bundled set of files constituting software. May be installed or uninstalled (counting as setup files)		Find-Package Get-Package Install-Package Save-Package Uninstall-Package

(Xumins, 2015. Introducing PackageManagement in Windows 10)
http://blogs.technet.com/b/packagemanagement/archive/2015/04/29/introducing-packagemanagement-in-windows-10.aspx

Provider operations

Get locally installed providers.

```
PS> Get-PackageProvider | ft -AutoSize

Name     Version          DynamicOptions
----     -------          --------------
Programs 10.0.10240.16384 {IncludeWindowsInstaller, IncludeSystemComponent}
msu      10.0.10240.16384 {}
msi      10.0.10240.16384 {AdditionalArguments}
PSModule 1.0.0.0          {PackageManagementProvider, Location, InstallUpdat...
NuGet    2.8.5.127        {Destination, SkipDependencies, ContinueOnFailure,...
```

(Bentley, n.d.) 2015-08-14 00:03
(Xumins, 2015. Introducing PackageManagement in Windows 10)
http://blogs.technet.com/b/packagemanagement/archive/2015/04/29/introducing-packagemanagement-in-windows-10.aspx

Source operations

Get locally installed sources.

```
PS> Get-PackageSource | ft -AutoSize

Name       ProviderName IsTrusted IsRegistered IsValidated Location
----       ------------ --------- ------------ ----------- --------
PSGallery  PSModule     False     True         False       https://www.power...
nuget.org  NuGet        False     True         False       https://www.nuget...
chocolatey Chocolatey   False     True         False       http://chocolatey...

PS> Get-PackageSource | ft Name, ProviderName, Location -AutoSize

Name       ProviderName Location
----       ------------ --------
PSGallery  PSModule     https://www.powershellgallery.com/api/v2/
nuget.org  NuGet        https://www.nuget.org/api/v2/
chocolatey Chocolatey   http://chocolatey.org/api/v2/
```

(Bentley, n.d.) 2015-08-14 00:10
(Xumins, 2015. Introducing PackageManagement in Windows 10)
http://blogs.technet.com/b/packagemanagement/archive/2015/04/29/introducing-packagemanagement-in-windows-10.aspx

Package operations

Discover remote package.

```
# Basic Example.
PS> Find-Package -Name PsReadline -ProviderName PSModule | ft -AutoSize

Name       Version  Source                                    Summary
----       -------  ------                                    -------
PSReadline 1.0.0.13 https://www.powershellgallery.com/api/v2/


# Find all versions.
PS> Find-Package -Name PsReadline -ProviderName PSModule -AllVersions | ft -AutoSize

Name       Version  Source                                    Summary
----       -------  ------                                    -------
PSReadline 1.0.0.10 https://www.powershellgallery.com/api/v2/
PSReadline 1.0.0.11 https://www.powershellgallery.com/api/v2/
PSReadline 1.0.0.12 https://www.powershellgallery.com/api/v2/
PSReadline 1.0.0.13 https://www.powershellgallery.com/api/v2/
PSReadline 1.0.0.8  https://www.powershellgallery.com/api/v2/
PSReadline 1.0.0.9  https://www.powershellgallery.com/api/v2/


# Same package, different providers (and therefore sources).
PS> Find-Package -Name NotePad2 -ProviderName NuGet | ft -AutoSize

Name     Version  Source    Summary
----     -------  ------    -------
notepad2 4.2.25.3 nuget.org A fast and light-weight Notepad-like text editor with syntax
highlighting.

PS> Find-Package -Name NotePad2 -ProviderName Chocolatey | ft -AutoSize

Name     Version  Source     Summary
----     -------  ------     -------
notepad2 4.2.25.3 chocolatey A fast and light-weight Notepad-like text editor with syntax
highlighting.
```

(Bentley, n.d.) 2015-08-14 00:10
(Xumins, 2015. Introducing PackageManagement in Windows 10)
http://blogs.technet.com/b/packagemanagement/archive/2015/04/29/introducing-packagemanagement-in-windows-10.aspx

Install package from remote source.

```
PS> Install-Package -Name NotePad2 -ProviderName Chocolatey | ft -AutoSize

Name     Version  Source     Summary
----     -------  ------     -------
notepad2 4.2.25.3 chocolatey A fast and light-weight Notepad-like text editor wit...

# Tangentially, the above command takes care of native notepad replacement. E.g.
# In windows explorer right click your text file and choose "Edit".
# Notepad2 will edit that file !
```

(Bentley, n.d.) 2015-08-14 00:55

Get a list of packages from one provider.

```
# Return a list of installed programs on windows with "Redist" in the name
PS> Get-Package -ProviderName Programs -Name Redist | sort name | ft name

Name
----
Microsoft Visual C++ 2012 Redistributable (x64) - 11.0.60610
Microsoft Visual C++ 2012 Redistributable (x64) - 11.0.61030
Microsoft Visual C++ 2012 Redistributable (x86) - 11.0.60610
Microsoft Visual C++ 2012 Redistributable (x86) - 11.0.61030
Microsoft Visual C++ 2013 Redistributable (x64) - 12.0.21005
Microsoft Visual C++ 2013 Redistributable (x86) - 12.0.21005
Microsoft Visual C++ 2015 RC Redistributable (x64) - 14.0.22816
Microsoft Visual C++ 2015 RC Redistributable (x86) - 14.0.22816
Microsoft Visual C++ 2015 Redistributable (x64) - 14.0.23506
Microsoft Visual C++ 2015 Redistributable (x86) - 14.0.23506
```

(Bentley, n.d.) 2016-04-12 10:45

Get an inventory of locally installed packages.

```
# Get a particular set of packages
# Here we use partial value matching. Wildcards forbidden.
PS> Get-Package Note | ft -AutoSize

Name                          Version  Source
----                          -------  ------
notepad2                      4.2.25.3 C:\Chocolatey\lib\notepad2.4.2.25.3\note...
Notepad2 (Notepad Replacement) 4.2.25


# Handy code for limiting the width of column output ...
PS> Get-Package Windows | Sort-Object Name | Format-Table ( `
@{Expression={$_.Name};Label="Name";width=50}, `
@{Expression={$_.Version};Label="Version";width=25}, `
@{Expression={$_.ProviderName};Label="ProviderName";width=25} `
)

Name                                               Version                  ProviderName
----                                               -------                  ------------
Behaviors SDK (Windows Phone) for Visual Studio... 12.0.51210.80            msi
Behaviors SDK (Windows) for Visual Studio 2013     12.0.51210.80            msi
Blend for Visual Studio SDK for Windows Phone 8.0  3.0.30924.0              msi
Cumulative Update for Windows 10 for x64-based ...                          msu
Cumulative Update for Windows 10 for x64-based ...                          msu


# Return all packages, Groupby ProviderName
PS> Get-Package | Sort-Object ProviderName, Name | Format-Table ( `
@{Expression={$_.Name};Label="Name";width=50}, `
@{Expression={$_.Version};Label="Version";width=25}, `
```

```
@{Expression={$_.ProviderName};Label="ProviderName";width=25} `
) `
-GroupBy ProviderName

   ProviderName: Chocolatey

Name                                           Version                 ProviderName
----                                           -------                 ------------
notepad2                                       4.2.25.3                Chocolatey


   ProviderName: msi

Name                                           Version                 ProviderName
----                                           -------                 ------------
 Tools for .Net 3.5                            3.11.50727              msi
3Dconnexion 3DxWare (x64)                      6.16.0                  msi
3Dconnexion Add-In for AutoCAD 2007 - 2010     4.5.1                   msi
...

   ProviderName: msu

Name                                           Version                 ProviderName
----                                           -------                 ------------
Cumulative Update for Windows 10 for x64-based ...                     msu
Cumulative Update for Windows 10 for x64-based ...                     msu
...

   ProviderName: Programs

Name                                           Version                 ProviderName
----                                           -------                 ------------
3Dconnexion 3DxSoftware (x64 Edition)          3.16.3                  Programs
Adobe AIR                                      16.0.0.273              Programs
Adobe Flash Player 18 NPAPI                    18.0.0.232              Programs
...
# Include a source column
PS> Get-Package Note | Sort-Object Name | Format-Table ( `
@{Expression={$_.Name};Label="Name";width=25}, `
@{Expression={$_.Version};Label="Version";width=11}, `
@{Expression={$_.ProviderName};Label="ProviderName";width=14}, `
@{Expression={$_.Source};Label="Source";width=35} `
)


Name                      Version     ProviderName   Source
----                      -------     ------------   ------
notepad2                  4.2.25.3    Chocolatey     C:\Chocolatey\lib\notepad2.4.2.2...
Notepad2 (Notepad Repl... 4.2.25      Programs
```

(Bentley, n.d.) 2015-08-14 01:30
Based on: StackExchange > Superuser > " How can I shrink the width of my columns in powershell on Windows 10?" > Answer By MrStatic, http://superuser.com/a/956308

Return installed files from Windows "Programs and Features", filtering by name.

```
PS > Get-Package -ProviderName msi -Name Java* | Sort name | Format-Table -Property Name,
Version, Source

Name                                Version     Source
----                                -------     ------
Java 8 Update 102                   8.0.1020.14 C:\Program Files (x86)\Java\jre1.8.0_102\
Java Auto Updater                   2.8.102.14
Java SE Development Kit 8 Update 102 8.0.1020.14 C:\Program Files (x86)\Java\jdk1.8.0_102\
```

Uninstall package.

```
PS> Uninstall-Package NotePad2 | ft -AutoSize

Name                          Version  Source
----                          -------  ------
notepad2                      4.2.25.3 C:\Chocolatey\lib\notepad2.4.2.25.3\...
```

```
Notepad2 (Notepad Replacement) 4.2.25

# At the date of testing Uninstall-Package only removes the package info from the
# Locally installed provide. The software remains installed in programs.
PS> Get-Package Note | Sort-Object Name | Format-Table ( `
@{Expression={$_.Name};Label="Name";width=25}, `
@{Expression={$_.Version};Label="Version";width=11}, `
@{Expression={$_.ProviderName};Label="ProviderName";width=14}, `
@{Expression={$_.Source};Label="Source";width=35} `

Name                      Version     ProviderName   Source
----                      -------     ------------   ------
Notepad2 (Notepad Repl... 4.2.25      Programs
```

(Bentley, n.d.) 2015-08-14 00:57

If you install a package via Powershell PackageManager (and a provider like chocolatey) then at Uninstall time you must:

1. Uninstall through Powershell PackageManager (with "Uninstall-Package"); and
2. Uninstall through Window's "Programs and Features".

Installation flakiness. If you intall from a provider/source like chocolatey and you:

- Uninstall the package manually through Window's "Programs and Features" then the chocolately package information remains ... so you can't use "install-package" to reinstall a now missing package;
- Uninstall the package through powershell PackageManager with "Uninstall-Package" then the software remains installed on the machine ... you can verify this by viewing it in Window's "Programs and Features".

(Bentley, n.d.) 2015-08-14 03:18

Visual Studio Package Manager Console

Visual Studio > View > Other Windows > Package Manager Console.

The Visual Studio Package Manager Console has access to a greater range of Cmdlets than native powershell. E.g. "Update-Package".

The Visual Studio Package Manager Console appears to be more about installing source files into existing .Net projects.

Powershell Module Management

Install a powershell module.

```
# Install PowerShell Community Extensions.
Install-Package -Name PSCX -ProviderName PSModule
```

(Bentley, n.d.) 2015-08-31 23:41

Creating .NET user interfaces

Powershell is remarkably powerful in being able to create .NET interfaces. See

(Microsoft, 2014. Scripting with Windows PowerShell) Windows PowerShell Tips, https://technet.microsoft.com/en-us/library/dn792466.aspx

Path (and Windows Environment)

Examine the Windows Path environment variable.

```
# See each individual path in the Path variable
PS> $env:Path -split ';'

# ... sorted
PS> $env:Path -split ';' | sort
```

(Bentley, n.d.) 2015-08-20 20:41

Get Path And File Name Parts

To ge the path of the currently running script use `$PSCommandPath`

```
# Test.ps1
$PSCommandPath

# Output
PS> .\Test.ps1
C:\Users\John\Documents\Sda\Code\Windows\PowerShell\Apps\Test.ps1
```

C:\Users\John\Documents\Sda\Code\Windows\PowerShell\Examples\Demo-PathAndFileNameParts.ps1

To get path and file name parts use Split-Path.

```
# Test.ps1
-------------------------------------------
# Full Path of the currently running script
$PSCommandPath
''

# Path and file name parts
Split-Path -Path $PSCommandPath -Qualifier
Split-Path -Path $PSCommandPath -NoQualifier
Split-Path -Path $PSCommandPath -Parent
Split-Path -Path $PSCommandPath -Leaf
Split-Path -Path $PSCommandPath -LeafBase
-------------------------------------------

# Output
PS> .\Test.ps1
C:\Users\John\Documents\Sda\Code\Windows\PowerShell\Apps\Test.ps1

C:
\Users\John\Documents\Sda\Code\Windows\PowerShell\Apps\Test.ps1
C:\Users\John\Documents\Sda\Code\Windows\PowerShell\Apps
Test.ps1
Test
```

C:\Users\John\Documents\Sda\Code\Windows\PowerShell\Examples\Demo-PathAndFileNameParts.ps1

Command timing or benchmarking

Measure the elapsed time of a command.

```
$sw = [Diagnostics.Stopwatch]::StartNew()
Get-ChildItem -Recurse | Format-Table
$sw.Stop()
$sw.Elapsed
```

Wait-Seconds

Wait-Seconds.

```
function Wait-Seconds([double] $Seconds) {
  # Pause the script for the specified $Seconds and show progress on the next line.
  # That is, with the progress line being overwritten with a percentage complete.
  # Based on code at:
  # Dullson. 2015. "PowerShell - Overwriting Line Written with Write-Host (Answer To)".
  # Stack Overflow. https://stackoverflow.com/questions/25846889/powershell-overwriting-
line-written-with-write-host. 2015-05-23.

  "Paused for $Seconds seconds."
  $StepSeconds = 1
  for ($i = $StepSeconds; $i -le $Seconds; $i++ )  {
    $PercentComplete = ($i/$Seconds) * 100
    Start-Sleep -Seconds $StepSeconds
    # Write-Progress -Activity "Pause Progress" -Status "$PercentComplete% Complete:" -
PercentComplete $PercentComplete;
    # The `r carriage return (without `n) is essential to overwrite the existing line
    Write-Host -NoNewline "`r$PercentComplete% complete."
  }
}
```

C:\Users\John\Documents\Sda\Code\Windows\PowerShell\Apps\xStopProcesses\xStopProcesses.ps1
Dullson. 2015. "PowerShell - Overwriting Line Written with Write-Host (Answer To)". Stack Overflow. https://stackoverflow.com/questions/25846889/powershell-overwriting-line-written-with-write-host . 2015-05-23.

Display message box

Display message box.

```
Add-Type -AssemblyName PresentationCore,PresentationFramework
$ButtonType = [System.Windows.MessageBoxButton]::OK
$MessageIcon = [System.Windows.MessageBoxImage]::Warning
$MessageBody = "Everything is cool"
$MessageTitle = "Something"

$Result =
[System.Windows.MessageBox]::Show($MessageBody,$MessageTitle,$ButtonType,$MessageIcon)

Write-Host "Your choice is $Result"
```

C:\Users\John\Documents\Sda\Code\Windows\PowerShell\Examples\Demo-ShowMessagebox.ps1

Rename a batch of files

Rename a batch of files.

```
# There is no need to filter before the pipe if our regex sufficiently filters after the
pipe. And altering the command string (e.g. if you copy the above command and now want to
use it to change the extension of '.xml' files) is no longer required in two places.
Get-ChildItem | Rename-Item -NewName { $_.Name -replace '\.txt$','.log' }

# To ensure the regex -replace operator matches only an extension at the end of the string,
include the regex end-of-string character "$".
PS> Get-ChildItem *.txt | Rename-Item -NewName { $_.Name -replace '\.txt$','.log' }

# From (Powershell 7.0.3)
PS> get-help rename-item -full

# ... the example is ...
PS> Get-ChildItem *.txt | Rename-Item -NewName { $ .name -Replace '.txt','.log' }

# ... That is, the <original> paramater in the -Replace regex string operator, '.txt': does
not escape the period with '\'; and does not ensure it is matching in the last position in
the string.
```

```
# So if we have files in our directory like ...

lorem01.txt
lorem02.txt.txt

# The help example will do to many replacements ...
PS> Get-ChildItem *.txt | Rename-Item -NewName { $_.Name -replace '\.txt','.log' } -whatif
... target ... lorem01.txt Destination: ... lorem01.log
... target ... lorem02.txt.txt Destination: ... lorem02.log.log

# By contrast our command with work as desired, only replacing the extension at the string
end ...
PS> Get-ChildItem *.txt | Rename-Item -NewName { $_.Name -replace '\.txt$','.log' } -whatif
... target ... lorem01.txt Destination: ... lorem01.log
... target ... lorem02.txt.txt Destination: ... lorem02.txt.log
```

https://docs.microsoft.com/en-us/powershell/module/microsoft.powershell.core/about/about_comparison_operators?view=powershell-7#replacement-operator
"How can I bulk rename files in PowerShell?", answer by dugas, https://stackoverflow.com/a/13382966/872154

References, Word

Anon., 2015. *Stackoverflow*. [Online] Available at: http://stackoverflow.com.

Babinec, K., (2013-Jun-13). *An Introduction to Error Handling in PowerShell*. [Online]
Available at: http://blogs.msdn.com/b/kebab/archive/2013/06/09/an-introduction-to-error-handling-in-powershell.aspx
[Accessed 2015-Aug-03].

Bentley, J., (2015). *dotNet Format Strings*. [Electronic Source]
Available at: \\ATLAS\Documents\Sda\Info\DotNet\KB\Quick Reference\Code\Operators And Special Characters\dotNet Format Strings.docx.

Bentley, J., (2015-Jul-29). *dotNet Framework Regular Expression Quick Reference*. [Electronic Source]
Available at: \\ATLAS\Documents\Sda\Info\DotNet\KB\Quick Reference\Code\Operators And Special Characters\dotNet Framework Regular Expression Quick Reference.docx.

Bentley, J., n.d. *Experimentally Verified.*

Microsoft, (2014-Aug-04). *Scripting with Windows PowerShell*. [Online]
Available at: https://technet.microsoft.com/library/bb978526.aspx
[Accessed 2015-Jul-02].

Microsoft, (2015)..*NET Framework Class Library*. [Online]
Available at: https://msdn.microsoft.com/en-us/library/gg145045%28v=vs.110%29.aspx
[Accessed 2015-Jul-15].

Microsoft, (2015). *Approved Verbs for Windows PowerShell Commands*. [Online]
Available at: https://msdn.microsoft.com/en-us/library/ms714428%28VS.85%29.aspx
[Accessed 2015-Jul-13].

Microsoft, (2015). *Core Modules in Windows PowerShell*. [Online]
Available at: https://technet.microsoft.com/en-us/library/mt156967.aspx
[Accessed 2015-Jul-06].

Microsoft, (2015). *WMI Reference*. [Online]
Available at: https://msdn.microsoft.com/en-us/library/aa394572%28v=vs.85%29.aspx
[Accessed 2015-July-10].

Sheppard, S., (2015). *Windows PowerShell command line syntax*. [Online]
Available at: http://ss64.com/ps/syntax.html
[Accessed 2015-Jul-06].

Wories, M., (2009-Jun-8). *PowerShell Tips & Tricks: Getting more detailed error information from PowerShell*. [Online]
Available at: http://blogs.msdn.com/b/mwories/archive/2009/06/08/powershell-tips-tricks-getting-more-detailed-error-information-from-powershell.aspx
[Accessed 2015-Aug-03].

Xumins, M.-, (2015-Apr-28). *Introducing PackageManagement in Windows 10*. [Online]
Available at: http://blogs.technet.com/b/packagemanagement/archive/2015/04/29/introducing-packagemanagement-in-windows-10.aspx
[Accessed 2015-Aug-13].

Document Licence