PowerShell Scripting :

A Shell is a user interface that gives access to various services of an Operating System. Windows PowerShell is a Shell Developed by Microsoft for the purpose of Task Automation & Configuration Management.

PowerShell versions: v1.0 to v5.0

PowerShell is almost Ten years old, and five different PowerShell versions are currently available.

PowerShell 2.0 and PowerShell 3.0 were major releases, with many new important features. PowerShell 4.0 was only released one year after PowerShell 3.0. Thus, we couldn't expect too many enhancements. Its main new feature is Desired State Configuration (DSC), a new platform for managing Windows services and their environment.

- $psversiontable – to know the Ps version

Shortcut keys

- F5 – Run script
- F8 – Run Selected
- F9 – Toggle point /Debug break point
- F10 – Step Over
- F11 – Step Into

PowerShell scripts (files suffixed by `.ps1`)
In PowerShell, administrative tasks are generally performed by *cmdlets* (pronounced *command-lets*).Cmdlets are specialized commands in the PowerShell environment that implement specific functions. Cmdlets follow a *Verb-Noun* naming pattern, such as *Get-ChildItem*, helping to make them self-descriptive. Sets of cmdlets may be combined into *scripts* & *executables*

PowerShell ISE

The Windows PowerShell Integrated Scripting Environment (ISE) is one of two hosts for the Windows PowerShell engine and language. With it you can write, run, and test scripts in ways that are not available in the Windows PowerShell Console.

Using Help in Powershell :

Displays information about Windows PowerShell commands and concepts. To get help for a Windows PowerShell command, type `Get-Help` followed by the command name, such as: `Get-Help Get-Process`. To get a list of all help topics on your system, type `Get-Help *However, starting in Windows PowerShell 3.0, the modules that come with the Windows operating system do not include help files. To download or update the help files for a module in Windows PowerShell 3.0, use the Update-Help cmdlet.

- Get-help
- Update-help

PowerShell Drives:

A *Windows PowerShell drive* is a data store location that you can access like a file system drive in Windows PowerShell. The Windows PowerShell providers create some drives for you, such as the file system drives (including C: and D:), the registry drives (HKCU: and HKLM:), and the certificate drive (Cert:), and you can create your own Windows PowerShell drives. These drives are very useful, but they are available only within Windows PowerShell. You cannot access them by using other Windows tools, such as File Explorer or Cmd.exe.

- Get-psdrive
- New-psdrive
- Remove-psdrive

Powershell Modules :

As the name implies, a *script module* is a file (.psm1) that contains any valid Windows PowerShell code. Script developers and administrators can use this type of module to create modules whose members include functions, variables, and more. At heart, a script module is simply a Windows PowerShell script with a different extension, which allows administrators to use import, export, and management functions on it.

- New-module
- Import-module
- Get-module
- Remove-module

Pipeline Concepts

1. Cmdlets work with objects
2. Piped from one command to the next
3. PowerShell displays remaining objects at the end of the pipeline
4. Objects may change in the pipeline
5. You get results only if there are objects in the pipeline

Write-Host vs. Write-Output

- Write-Host skips the pipeline
  - Writes to the console or hosting application
  - Can't be redirected
  - NO OBJECTS
- Write-Output writes the pipeline – Preferred behavior
  - Commands implicitly write pipelined output
  - Test by piping expression to Get-Member

Note: Watch for –Passthru

  - –Some commands don't write to the pipeline by default
  - PS C:\> Stop-Service wuauserv -passthru

Using Variables

- A variable is a place holder or container for a PowerShell object
  – It "is" whatever it contains
  – Can be a collection of objects
- Create by assignment
  – PS C:\> $x = 123
  – PS C:\> $scripts = dir c:\scripts\*.ps1
- Use variables as placeholders
  – PS C:\> $x*2   =246
  – PS C:\> $scripts.count = 232
- Variables are "point in time"
- Variable names contain letters, numbers, and _.
- Avoid spaces
- Give variables meaningful names

Subexpressions

- Use parentheses to control pipelined execution
  PS C:\> get-service wuauserv –computer (get-content c:\work\computers.txt)
- You can reference object properties as subexpressions
  PS C:\> $svc = get-service browser
  PS C:\> "The $($svc.displayname) service is $($svc.status)"

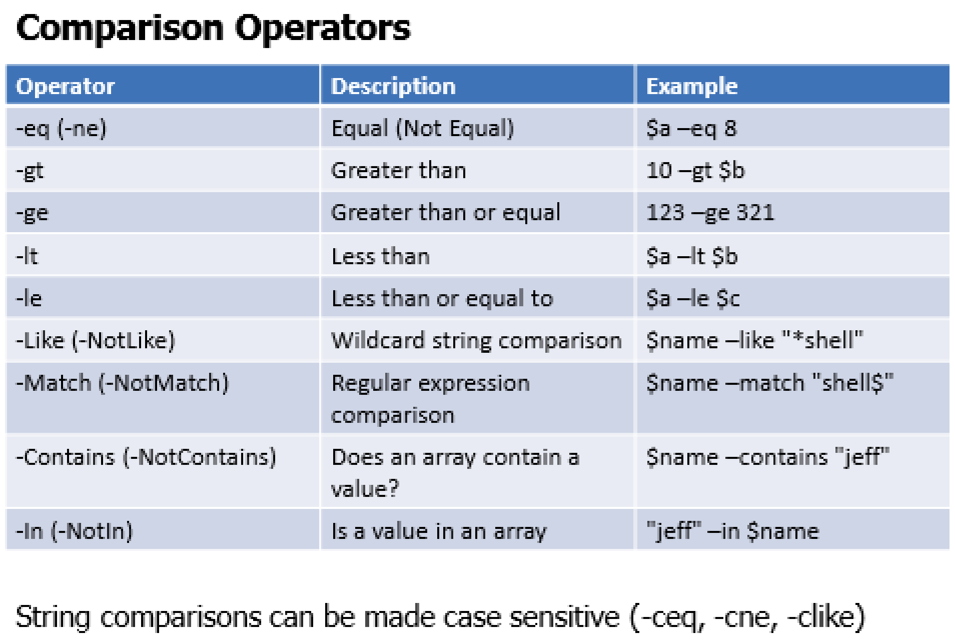
Redirecting Output

- Using Out-File
  – Use –Append to add to a file
- Tee-Object
  – Send to pipeline and a file
  – Send to pipeline and a variable
- Out* cmdlets should be at the end of your expression
  --c:\> dir –file -hidden | out-file –filepath c:\work\rootfiles.txt

- Write to file >
- Append to file >>
- Merge to file >&

Operators :

Comparision Operator
"How does something compare to something else?"
Expression must come out as True or False
PowerShell is not cAsEsEnsitIve unless you make it

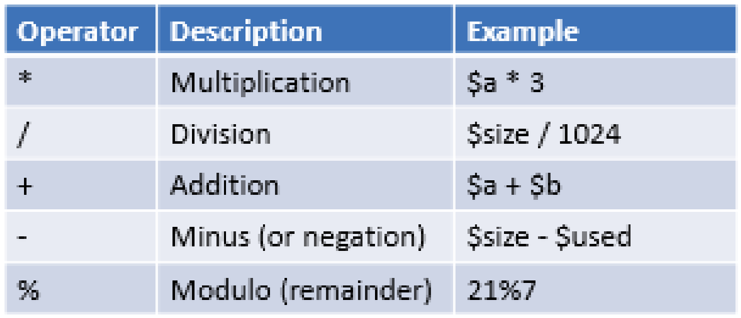


Comparison Operators

Operator	Description	Example
-eq (-ne)	Equal (Not Equal)	$a –eq 8
-gt	Greater than	10 –gt $b
-ge	Greater than or equal	123 –ge 321
-lt	Less than	$a –lt $b
-le	Less than or equal to	$a –le $c
-Like (-NotLike)	Wildcard string comparison	$name –like "*shell"
-Match (-NotMatch)	Regular expression comparison	$name –match "shell$"
-Contains (-NotContains)	Does an array contain a value?	$name –contains "jeff"
-In (-NotIn)	Is a value in an array	"jeff" –in $name

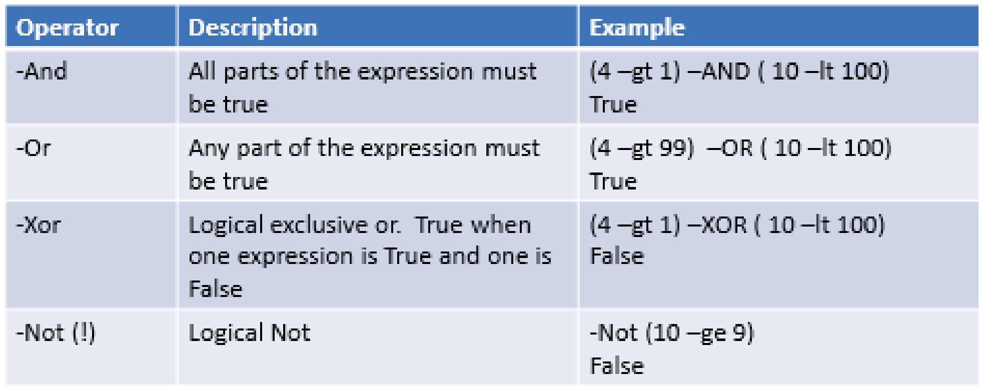
String comparisons can be made case sensitive (-ceq, -cne, -clike)

Arithmetic Operators :



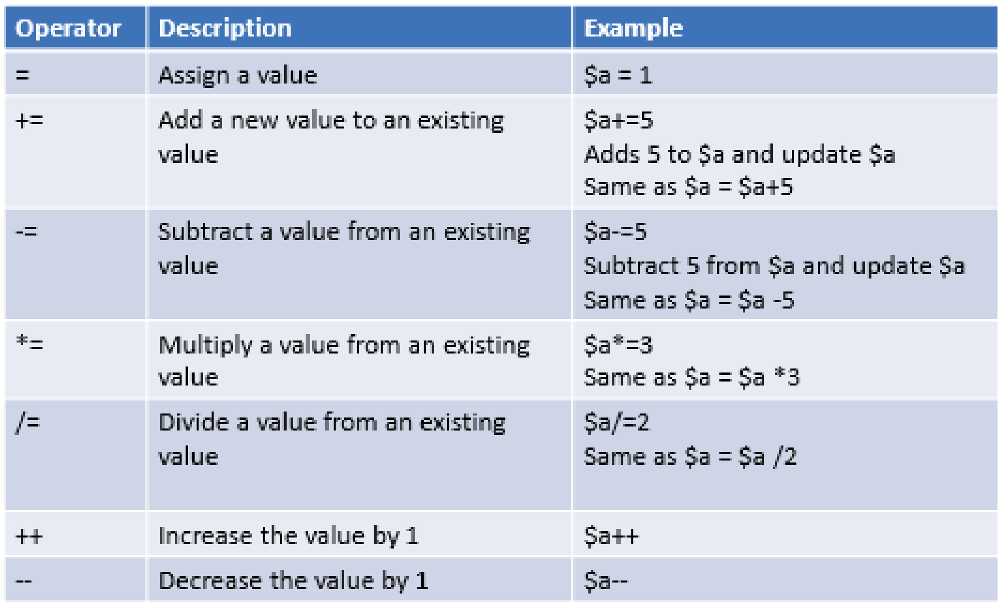
Operator	Description	Example
*	Multiplication	$a * 3
/	Division	$size / 1024
+	Addition	$a + $b
-	Minus (or negation)	$size - $used
%	Modulo (remainder)	21%7

Logical Operators:



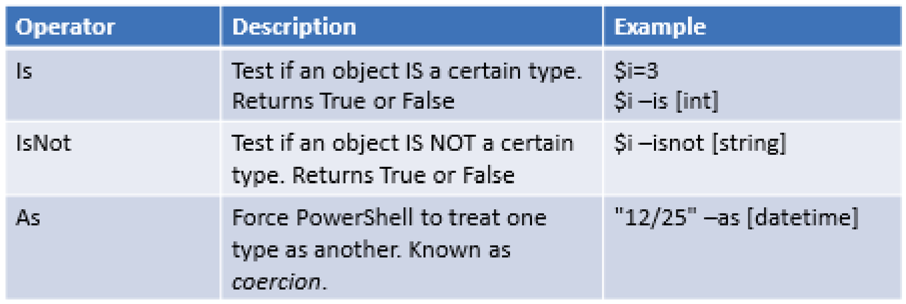
Operator	Description	Example
-And	All parts of the expression must be true	(4 –gt 1) –AND (10 –lt 100) True
-Or	Any part of the expression must be true	(4 –gt 99) –OR (10 –lt 100) True
-Xor	Logical exclusive or. True when one expression is True and one is False	(4 –gt 1) –XOR (10 –lt 100) False
-Not (!)	Logical Not	-Not (10 –ge 9) False

Assignment Operators :



Operator	Description	Example
=	Assign a value	$a = 1
+=	Add a new value to an existing value	$a+=5 Adds 5 to $a and update $a Same as $a = $a+5
-=	Subtract a value from an existing value	$a-=5 Subtract 5 from $a and update $a Same as $a = $a -5
*=	Multiply a value from an existing value	$a*=3 Same as $a = $a *3
/=	Divide a value from an existing value	$a/=2 Same as $a = $a /2
++	Increase the value by 1	$a++
--	Decrease the value by 1	$a--

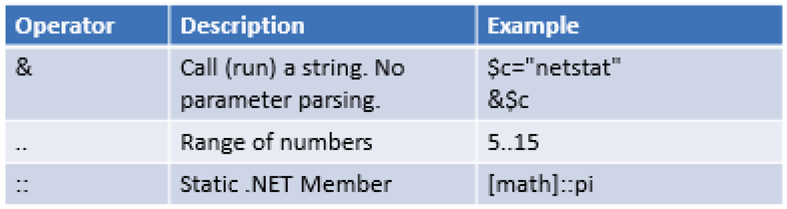
Type Operators :



Operator	Description	Example
Is	Test if an object IS a certain type. Returns True or False	$i=3 $i –is [int]
IsNot	Test if an object IS NOT a certain type. Returns True or False	$i –isnot [string]
As	Force PowerShell to treat one type as another. Known as *coercion*.	"12/25" –as [datetime]

Common Types: [Int32] [Double] [String] [DateTime]

Special Operator:



Operator	Description	Example
&	Call (run) a string. No parameter parsing.	$c="netstat" &$c
..	Range of numbers	5..15
::	Static .NET Member	[math]::pi

•Number "Shortcuts" – xKB, xMB, xGB, xTB, xPB – X is the number of units (e.g. 5MB)

- PS C:\> 3GB 3221225472
- PS C:\> $size/1MB 1224.0986328125

Split and Join

•Split

- <string> -split <delimiter>
- Default delimiter is the space
- Can split on a regular expression pattern
- Creates an array of strings – Split into substrings
- PS C:\> $vowels = a:e:i:o:u $data = $vowels –split ":"

•Join

- -join <string[]>
- <string[]> -join <delimiter>
- Default delimiter is nothing
- PS C:\> "PowerShell","3.0","is","awesome" –join " "

====================End of Chapter1===================================