### Useful Commands

<table>
<thead>
<tr>
<th>Command</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>Update-Help</td>
<td>Downloads and installs newest help files</td>
</tr>
<tr>
<td>Get-Help</td>
<td>Displays information about commands and concepts</td>
</tr>
<tr>
<td>Get-Command</td>
<td>Gets all commands</td>
</tr>
<tr>
<td>Get-Member</td>
<td>Gets the properties and methods of objects</td>
</tr>
<tr>
<td>Get-Module</td>
<td>Gets the modules that have been imported or that can be imported into the current session</td>
</tr>
</tbody>
</table>

### Operators

#### Assignment Operators
- , =, +=, -=, *, /, %, %*, +=, -=
  - Assigns one or more values to a variable

#### Comparison Operators
- eq, ne
  - Equal, not equal
- gt, ge
  - Greater than, greater than or equal to
- lt, le
  - Less than, less than or equal to
- replace
  - Changes the specified elements of a value

#### “abcde” -replace “bc”, “TEST”

#### Match Operators
- match, -notmatch
  - Regular expression match
- like, -notlike
  - Wildcard matching
- contains, -notcontains
  - Returns TRUE if the scalar value on its right is contained in the array on its left

#### Range Operator
1..10

#### “abcdef” -split “de”

#### “abc”,”def”,”ghi” -join “,”

#### Type Evaluator
- is, -isnot
  - Type evaluator (Boolean). Tells whether an object is an instance of a specified .NET Framework type.

#### 42 -is [int]

#### Type Converter
- as
  - Type converter. Tries to convert the input object to the specified .NET Framework type.

#### 1..10 | foreach {$_ * 5}

#### Bitwise Operators
- band
  - Bitwise AND
- bor
  - Bitwise OR (inclusive)
- bxor
  - Bitwise OR (exclusive)
- bnot
  - Bitwise NOT
- shl, -shr
  - Bitwise shift operators. Bit shift left, bit shift right (arithmetic for signed, logical for unsigned values)

#### Other Operators
- Split
  - Splits a string
- “abcdefg” -split “de”
- join
  - Joins multiple strings
- “abc”,”def”,”ghi” -join “,”

#### Subexpression Operator
$() ( )

#### Array Subexpression Operator
$a = "Get-Process" & $a

#### Logical Operators
- and, -or, -xor, -not
  - Connect expressions and statements, allowing you to test for multiple conditions

#### Redirection Operators
- >, >>
  - The redirection operators enable you to send particular types of output (success, error, warning, verbose, and debug) to files and to the success output stream.

#### Output Streams
- * All output
  - 1 Success output
  - 2 Errors
  - 3 Warning messages
  - 4 Verbose output
  - 5 Debug messages

#### Dot-Sourcing Operator
$sb = { Get-Process | Select –First 2 } & $sb

#### Do-Something
- 3> warning.txt
- Append verbose.txt with the verbose output
- 4>> verbose.txt
- # Writes debug output to the output stream
- 5>&1
- # Redirects all streams to out.txt
- * > out.txt
### Arrays

- "a", "b", "c"  
  Array of strings
- 1,2,3  
  Array of integers
- @()  
  Empty array
- @(2)  
  Array of one element
- @((2), (1), (3))  
  Array within array
- @((2), (3), (4), "hi")  
  Array of one element
- @([5])  
  Sixth element of array*
- @([2..20])  
  Returns elements 3 thru 21
- @([-1])  
  Returns the last array element
- @([-3..-1])  
  Displays the last three elements of the array
- @([1..4..9])  
  Displays the elements at index positions 1, 4, and 6 through 9
- @(Get-Process)  
  Forces the result to an array using the array sub-expression operator
- @([arr][1..10])  
  Reverses an array
- @([arr].length-1..0]  
  Adds to an existing value of the second array item (increases the value of the element)
- @([arr][0, 1 + 3..6])  
  Creates a new array based on selected elements of an existing array
- @([arr] + $b)  
  Combines two arrays into a single array, use the plus operator (+)

*Arrays are zero-based

### Associative Arrays (Hash tables)

- $hash = @{ }  
  Creates empty hash table
- @{foo=1; bar='value2'}  
  Creates and initialize a hash table
- @(ordered){a=1; b=2; c=3}  
  Creates an ordered dictionary
- $hash.key1 = 1  
  Assigns 1 to key key1

### Comments

- Comments can be single-line or multi-line.
- Single-line comments start with `#`
- Multi-line comments start with `#`
- Block comments are enclosed with `@`.

### Variables

- Format: ${scope:name} or ${ananyname} or ${any path}
- Example: $path = "C:\Windows\System32"
- $processes = Get-Process
- $local: a = 1 # visible everywhere
- $local: a = 1 # defined in this scope and visible to children
- $private: a = 1 # same as local but invisible to child scopes
- $script: a = 1 # visible to everything is this script
- Using scope indicates a local variable in remote commands and with Start-Job
  
### Methods

- Methods can be called on objects.
  
```powershell
$a = "This is a string"
$a | Get-Member -MemberType Method
$a.ToUpper()
```
<table>
<thead>
<tr>
<th>Windows PowerShell Preference Variables (not exhaustive)</th>
</tr>
</thead>
<tbody>
<tr>
<td><code>$ConfirmPreference</code></td>
</tr>
<tr>
<td><code>$VerbosePreference</code></td>
</tr>
<tr>
<td><code>$WarningPreference</code></td>
</tr>
<tr>
<td><code>$WhatIfPreference</code></td>
</tr>
</tbody>
</table>

<table>
<thead>
<tr>
<th>Collection Filtering</th>
</tr>
</thead>
<tbody>
<tr>
<td>Collection filtering by using a method syntax is supported.</td>
</tr>
<tr>
<td>$Services = Get-Service</td>
</tr>
<tr>
<td>$Services.Where($_.Status -eq 'Stopped'), 'First', 3</td>
</tr>
<tr>
<td>$Services.ForEach($_.NameToUpper())</td>
</tr>
</tbody>
</table>
## WINDOWS POWERSHELL LEARNING RESOURCES

### Microsoft Resources

**Scripting with Windows PowerShell**  

**Windows PowerShell Team Blog**  
http://blogs.msdn.com/Powershell

**Microsoft Script Center**  
http://technet.microsoft.com/scriptcenter/default

**Windows PowerShell Forum**  

**Hey, Scripting Guy! Blog**  
http://blogs.technet.com/b/heyscriptingguy/

**Windows PowerShell Survival Guide**  

**Windows PowerShell ISE Add-on Tools**  

**Windows PowerShell Customer Connection: Submit bugs and feature requests**  
https://connect.microsoft.com/powershell

**Report Windows PowerShell documentation bugs by email**  
write-help@microsoft.com

### Community Resources

**PowerShell Code Repository**  
http://poshcode.org

**PowerShell.com Community**  
http://powershell.com

**PowerShell.org Community**  
http://powershell.org

**PowerGUI Community**  
http://en.community.dell.com/techcenter/powergui/

**The PowerShell Community Toolbar**  
http://powershell.ourtoolbar.com/

**PowerScripting Podcast**  
http://powerscripting.net

**PowerShell Magazine**  
http://powershellmagazine.com

irc.freenode.net #PowerShell

### Free eBooks and Guides

**Mastering PowerShell, Second Edition - Dr. Tobias Weltner**  

**PowerShell.org Free eBooks**  
http://powershell.org/wp/ebooks/

**Effective Windows PowerShell - Keith Hill**  
http://rkeithhill.wordpress.com/2009/03/08/effective-windows-powershell-the-free-ebook/

### Popular Community Projects

**PowerShell Community Extensions (PSCX)**  
http://pscx.codeplex.com/

**PSReadLine - A bash inspired readline implementation for PowerShell**  
https://github.com/Izybkr/PSReadLine

**TabExpansionPlusPlus - PowerShell module to improve tab expansion and Intellisense**  
https://github.com/Izybkr/TabExpansionPlusPlus

**PowerSploit - A PowerShell Post-Exploitation Framework**  
https://github.com/mattifestation/PowerSploit

**PoshSec - PowerShell module focused on security in the Windows environment**  
https://github.com/PoshSec/PoshSec

**Posh-SecMod - PowerShell module with security-related cmdlets**  
https://github.com/darkoperator/Posh-SecMod

**Pester - PowerShell BDD-style testing framework**  
https://github.com/pester/Pester