Testing, Loops and WMI
Summer 2022

Powershell

©DENNIS SIMPSON 2013-2022

Testing - if

- To test things, we can use the if statement
- We have one or more expressions to evaluate inside parentheses
- Multiple expressions can be used and prioritized with additional parentheses
- We have a script block to execute inside braces
- We can extend the test using elseif and else
- help about_if

©DENNIS SIMPSON 2013-2022

if Example

```
if ( $ConSeats -gt $LibSeats) {
 "Libs are Mad, bro"
}
elseif ( $LibSeats -gt $ConSeats ) {
 "Cons are mad, bro"
}
else {
 "Nobody happy, everyone mad"
}
```

- help about_comparison_operators

©DENNIS SIMPSON 2013-2022

Testing - Switch

- Switch is used for testing when you are executing one or more script blocks out of a group of script blocks based on a value or collection of values
- When you are testing a collection, matching script blocks are executed separately for each object in the collection
- break (terminate the switch) and continue (jump to the end of the script block) are available in the script blocks

©DENNIS SIMPSON 2013-2022

Switch Example

```
switch ( $myvar ) {
    0 { "myvar had a zero in it";continue }
    32 { "myvar had a 32 in it";continue }
    "rad" { "myvar was like, totally rad";continue }
    $yourvar { "Cool! myvar had the same guts as yourvar!";continue }
    {($_ -is [datetime]) -and ($_.dayofweek -lt $yourvar.dayofweek)} { "Rats. myvar's someproperty was less than yourvar's someproperty. You win.";continue }
    default { "I dunno about you, but myvar had something in it I didn't expect and it freaked me out" }
}
```

- help about_switch

©DENNIS SIMPSON 2013-2022

Working With Bitfields Switch Example

- When you are working with complex objects, data is sometimes encoded into bitfields
- This example demonstrates testing bit values to produce human readable output

FILE: printers.ps1

```
Get-WmiObject -class win32_printer |
   select name,
    @{n="Default?";e={if($_.attributes -band 4){$attr="default"};$attr}},
    @{n="Shared?";e={if($_.attributes -band 8){$attr="shared"};$attr}},
    @{n="Status";e={switch($_.printerstatus){1{$stat="other"}
                                           2{$stat="unknown"}
                                           3{$stat="idle"}
                                           4{$stat="printing"}
                                           5{$stat="warming up"}
                                           6{$stat="stopped printing"}
                                           7{$stat="offline"}};
                                           $stat}} |
   ft -AutoSize
```

©DENNIS SIMPSON 2013-2022

Looping On A Condition

- While and Until can be used to repeat a script block based on the result of an expression
- Putting Do at the start of a script block and While or Until after the end of it causes the script block to be run once before the condition is evaluated
- Until cannot be used without Do, but While can

```
while ($var -lt 5) {$var++ ; $var}

do {$var++;$var} while ($var -lt 5)

do {$var--;$var} until ($var -gt 1)
```

©DENNIS SIMPSON 2013-2022

While Examples

```
while ( $intf_speed -lt $minToMakeMeHappy ) { change-providers }

while ( ! $forgiven ) { buy-flowers }

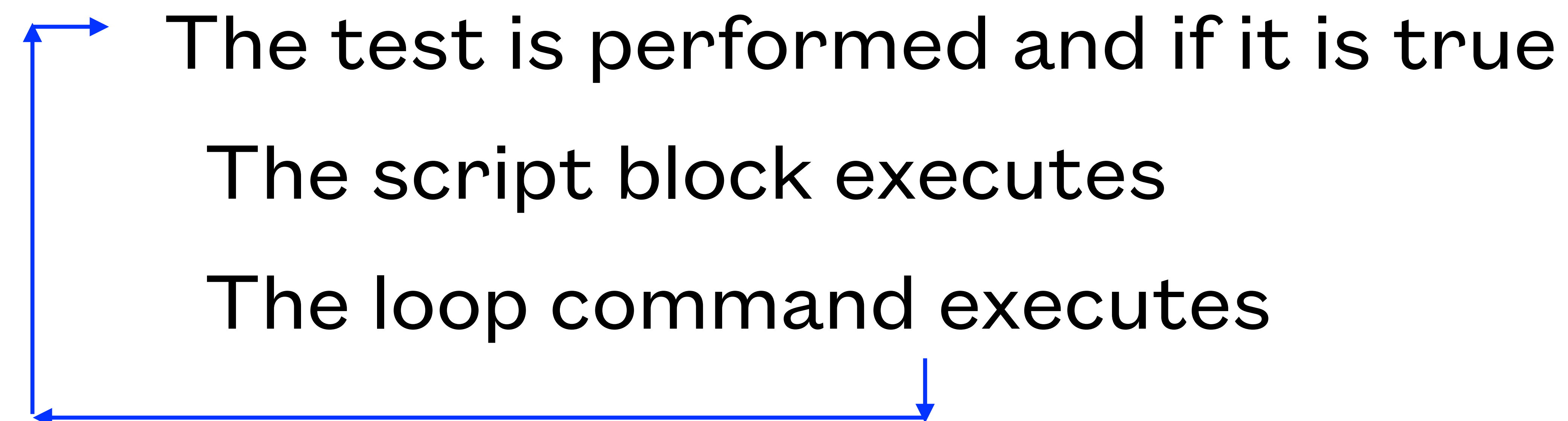
do {
  $annoyed = read-host -prompt “Are you annoyed yet [y/N]?”
} while ( $annoyed -notlike "y*" )

$chocolates = 6
while ( $chocolates -gt 0 ) {
  "Yum!" ; $chocolates--
  sleep 2
}
```

©DENNIS SIMPSON 2013-2022

For/Foreach

- foreach is used to execute a script once for each object in a collection
- for is used when you have an initial command, a test, and a loop command to perform



©DENNIS SIMPSON 2013-2022

For/Foreach Examples

- ```
  foreach -inputobject $collection {
    "The current object looks kinda like a " + $_.gettype().name
  }
  ```

- ```
  $objects | foreach-object {
                    "Wow, I got a " + $_ + "from the pipeline!" }
  ```

- ```
  for ( $counter = 0; $doghappy -ne $true; $counter++ ) {
    pet-dog
    feed-dog
  }
  "Dog requires level $counter attention to be happy"
  ```

©DENNIS SIMPSON 2013-2022
  ```

Foreach Example

```
$totalcapacity = 0
get-wmiobject -class win32_physicalmemory |
foreach {
        new-object -TypeName psobject -Property @{
            Manufacturer = $_.manufacturer
            "Speed(MHz)" = $_.speed
            "Size(MB)" = $_.capacity/1mb
            Bank = $_.banklabel
            Slot = $_.devicelocator
        }
        $totalcapacity += $_.capacity/1mb
} |
ft -auto Manufacturer, "Size(MB)", "Speed(MHz)", Bank, Slot
"Total RAM: ${totalcapacity}MB "
```

©DENNIS SIMPSON 2013-2022

Working Over The Network

- Powershell can run cmdlets over the network, executing them on remote hosts
- The remote host must enable remote access, and it only works between 2 computers running Windows
- The -ComputerName parameter is used to specify the remote computer to execute the cmdlet on
- Alternately, you can use psexec to remotely execute simple commands on remote machines
- See https://4sysops.com/archives/psexec-vs-the-powershell-remoting-cmdlets-invoke-command-and-enter-pssession/ for more information

©DENNIS SIMPSON 2013-2022

Get-WMIObject

- Get-WMIObject retrieves many types of system information objects, gwmi is an alias for get-wmiobject
- gwmi -list shows a list of the retrievable objects, add a word to the command to limit the output based on the class name, * is allowed in the word
  e.g gwmi -list *adapter*
- WMIExplorer and the online resources from blackboard are also good places to discover useful WMI classes
- WMI is widely used, but deprecated in favour of CIM, which uses the Get-CIMInstance cmdlet and the same WMI classes as well as some CIM versions of those classes

©DENNIS SIMPSON 2013-2022

Some Interesting WMI Classes

- win32_computersystem
  win32_operatingsystem
  win32_bios
- win32_processor
  win32_cachememory
  win32_physicalmemory
- win32_logicaldisk
  win32_diskdrive
  win32_diskpartition
- win32_videocontroller
  win32_desktopmonitor
- win32_networkadapter
  win32_networkadapterconfiguration
- win32_printer
- win32_usbcontrollerdevice

©DENNIS SIMPSON 2013-2022

Finding Related WMI Objects

- WMI objects have a GetRelated() method to find related WMI objects for the same device or resource as the one you already have
- Use gwmi -class someclassname|% {$_.getrelated().__CLASS} to see what related objects exist for someclassname
- You can then use new-object or select-object to build objects that use properties and methods from multiple WMI objects

```
$adapters = Get-WmiObject Win32_NetworkAdapter
$filteredadapters = $adapters | where-object {$_.adaptertype -match "ethernet" -and $_.netenabled -eq $true}
$myNetworkObjects = $filteredadapters |
                    Foreach { $adapter = $_;
                          $nac = $adapter.GetRelated("Win32_NetworkAdapterConfiguration");
                          New-Object PSObject -Property @{Name=$adapter.name;
                                                    ConnectionName=$adapter.netconnectionid;
                                                    IPAddress=$nac.ipaddress;
                                                    Gateway=$nac.defaultipgateway;
                                                    "Speed(MBps)" = $adapter.speed / 1000000
                                                   }
                        }
$myNetworkObjects | format-table Name, ConnectionName, IPAddress, Gateway, "Speed(Mbps)"
```

©DENNIS SIMPSON 2013-2022

Finding Related CIM Objects

- You can use Get-CIMAssociatedInstance to find other CIM class objects for the same device or resource as the one you already have (e.g. Get-CIMInstance CIM_LogicalDisk | Get-CIMAssociatedInstance -ResultClassName Win32_DiskPartition)
- Use Get-CIMInstance somecimobject | Get-CIMAssociatedInstance |% {$_.CreationClassName} to get a list of the related classes for somecimobject

```
$adapters = Get-CIMInstance CIM_NetworkAdapter
$filteredadapters = $adapters | where-object {$_.adaptertype -match "ethernet" -and $_.netenabled -eq $true}
$myNetworkObjects = $filteredadapters |
                    foreach { $adapter = $_;
                              $nac = $adapter | Get-CIMASsociatedInstance -resultclassname Win32_NetworkAdapterConfiguration;
                              New-Object PSObject -Property @{Name=$adapter.name;
                                                              IPAddress=$nac.ipaddress;
                                                              Gateway=$nac.defaultipgateway;
                                                              ConnectionName=$adapter.netconnectionid;
                                                              "Speed(Mbps)"=$adapter.speed / 1000000
                                                             }
                            }
$myNetworkObjects | format-table Name, ConnectionName, IPAddress, Gateway, "Speed(Mbps)"
```

©DENNIS SIMPSON 2013-2022

Lab 4 - Loops and WMI/CIM

©DENNIS SIMPSON 2013-2022