Powershell

Testing, Loops, Modules, More WMI
COMP2101 Fall 2017

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
- about_if

if Example

```
if ( $gob -eq $smacked) {
 "Gob was smacked"
}
elseif ( $LiberalSeats -gt $PCSeats ) {
 "Cons are mad, bro"
}
else {
 "No gobs smacked and cons are not happy"
}
```

- about_comparison_operators

Testing - Switch

- Switch is used for testing if you are executing one or more script blocks out of a group of script blocks based on a value or collection of values
- If 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

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" }

}
```

- about_switch

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
```

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

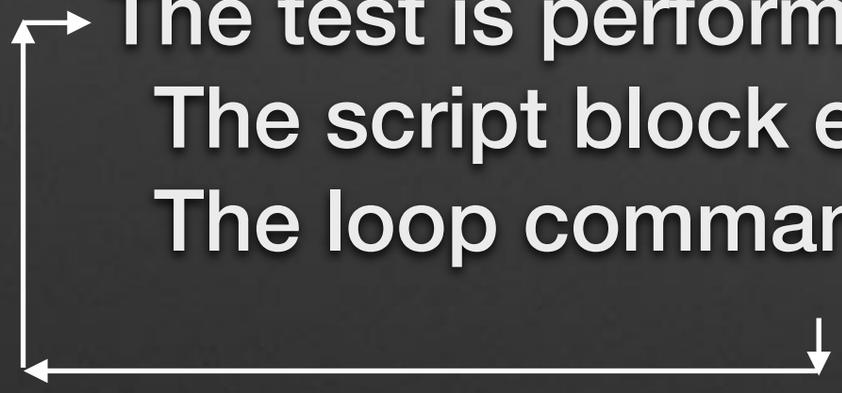
```
while ($var -lt 5) {$var++ ; $var}

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

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

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
}
```

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
  The initial command executes
  The test is performed and if it is true
    The script block executes
    The loop command executes



For/Foreach Examples

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

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

- for ($counter = 0; $doghappy -ne $true; $counter++) {
  pet-dog
  feed-dog
  }

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 "
```

Working Over The Network

- Powershell can run cmdlets over the network, executing them on remote hosts
- The remote host must enable remote access
- 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

Finding Related WMI Objects

- WMI objects have a GetRelated() method
- You can use it to find other WMI class objects for the same device or resource as the one you already have
- You can then use new-object or similar cmdlets to build objects that use properties and methods from both of the WMI objects

```
Get-WmiObject Win32_NetworkAdapter | # start with all network adapters
 ? adaptertype -match "ethernet" | # trim the collection to just ethernet adapters
 Foreach-Object { # loop through all ethernet adapters running a script block on each one
  $nac = $_.GetRelated("Win32_NetworkAdapterConfiguration") # get the related config object
  # make a new object from the original pair of related objects
  New-Object PSObject -Property @{name=$_.name
                        ipaddress=$nac.ipaddress|where-object {($_ -is [string]) -and ($_.indexof(".") -gt 0)}
                        ipgateway=$nac.defaultipgateway|where-object {($_ -is [string]) -and ($_.indexof(".") -gt 0)}
  }
 } | # end of script block to run on each adapter
 format-table name, ipaddress, ipgateway -AutoSize # format the object collection for display
```

Modules

- A module is at the minimum a collection of functions stored in a file with a .psm1 extension
- If you put the module file in $env:HOMEPATH/Documents/WindowsPowerShell/Modules/ModuleFileNameWithoutExtension/ ($PSModulePath), it will be automatically imported
- get-module -listavailable can be used to show modules not yet imported
- get-command -module modulename can be used to see what commands are in a module
- remove-module modulename can be used to remove a module from memory (e.g. you update the module file and want it to be imported again)
- Beware of name conflicts when creating modules, use common verbs whenever possible
- See https://msdn.microsoft.com/en-us/library/dd878340(v=vs.85).aspx for more information on creating modules

Lecture Videos

- There are lecture videos for this presentation on my youtube channel
- I have put together a playlist for you to view them in more or less the order the topics appear in the presentation slides

- https://www.youtube.com/playlist?list=PLIptG-28ZUJVG3k6Dkneqtq4gOdV4QzIb