Windows PowerShell 4.0 For the IT Professional - Part 1

Module15: Flow Control

Student Lab Manual

Version 2.0

Conditions and Terms of Use

Microsoft Confidential - For Internal Use Only

Contents

LAB 15: FLOW CONTROL 5

EXERCISE 15.1: LOOPING 6
Task 15.1.1: Looping – While 6
Task 15.1.2: Looping – Do While 6
Task 15.1.3: Looping – Do Until 7
Task 15.1.4: Looping – ForEach 7
Task 15.1.5: Looping – For 8
EXERCISE 15.2: BRANCHING 10
Task 15.2.1: Branching – If 10
Task 15.2.2: Branching – If Else 11
Task 15.2.3: Branching – Switch (simple) 11
Task 15.2.4: Branching – Switch (case sensitive) 12
Task 15.2.5: Branching – Switch (wildcards) 13
Task 15.2.6: Branching – Switch (Regular Expression) 13
Task 15.2.7: Branching – Switch (input from filename) 14
Task 15.2.8: Branching – Switch (and the break statement) 14
EXERCISE 15.3: BREAK AND CONTINUE KEYWORDS 16
Task 15.3.1: Flow Control – Break/Continue 16
Task 15.3.2: Flow Control – Exit 16

Lab 15: Flow Control

Introduction

Flow control enables decision-making within functions and scripts.

Objectives

After completing this lab, you will be able to:

- Understand the choices for the various looping constructs.
- Understand the use and capabilities of If and Switch
- Understand and use flow control keywords

Prerequisites

Start all VMs provided for the workshop labs.

Logon to WIN8-WS as:

Username: **Contoso\Administrator**

Password: **PowerShell4**

Estimated time to complete this lab

60 minutes

NOTE: These exercises use many Windows PowerShell commands. You can type these commands into the Windows PowerShell Integrated Scripting Environment (ISE) or the Windows PowerShell console. For some exercises, you can load pre-typed lab files into the Windows PowerShell ISE, allowing you to select and execute individual commands. Each lab has its own folder under **C:\PShell\Labs** on **WIN8-WS**.

Some exercises in this workshop may require running the Windows PowerShell console or ISE as an elevated user (Run as Administrator).

We recommend that you connect to the virtual machines (VMs) for these labs through Remote Desktop rather than connecting through the Hyper-V console. This allows you to use copy and paste between VMs and the host machine. If you are using the online hosted labs, then you are already using Remote Desktop.

Exercise 15.1: Looping

Introduction

Windows PowerShell includes a number of statements for stepping through a series of values in a collection – known as iterating.

Objectives

After completing this exercise, you will be able to:

- Understand the looping statements' syntax.
- Understand the choices for the various looping constructs.

Task 15.1.1: Looping – While

1. In the Windows PowerShell ISE, open the following file:
   C:\Pshell\Labs\Lab_15\Looping-While.ps1

```
$a=0

While ($a -lt 10)
{
   $a
   $a++
}
```

2. Highlight the While … line, press <F9>, and the line will turn red.
3. Execute the script by pressing <F5>. The next line to be executed will turn yellow.
4. Press <F11> to step through each line of the script.
5. At each iteration hover your mouse over the $a variable.
6. Note the changing values. Note the value of $a at the first iteration.
7. When is the evaluation that controls the loop occurring?

Do not stress that the student is using a "debugger". It is just the student stepping along. Nothing to be afraid of! If anybody wants more, suggest they attend Part 2.

Before we enter the script-block. If the evaluation is false, then the script-block is never executed.

Task 15.1.2: Looping – Do While

1. In the ISE, open the following file:
   C:\Pshell\Labs\Lab_15\Looping-DoWhile.ps1

```
$a=0

Do {$a}
While ($a++ -le 5)
```

2. Highlight the Do… line and press <F9>
3. Execute the script by pressing <F5>. The next line to be executed will turn yellow.
4. Press <F11> to step through each line of the script and hover your mouse over the $a variable on each iteration.
5. Note the changing values. Note the value of $a, at the first iteration.
6. When is the evaluation that controls the loop occurring?

After the script-lock has been executed at least once!

Task 15.1.3: Looping – Do Until

1. Open and execute the following script within the Windows PowerShell ISE script pane. C:\Pshell\Labs\Lab_15\Looping-DoUntil.ps1

```
$MinimumLength = (Get-ADdefaultDomainPasswordPolicy).MinPasswordLength

Do
{
    write-host "Password must be at least $MinimumLength characters long"
    $pwd = Read-Host "Enter a Password" -AsSecureString
}
Until ($pwd.Length -GE $MinimumLength)
Write-Host “Password Accepted”
```

2. Highlight the opening curly brace on the line after the Do keyword and press <F9> to set a breakpoint.
3. Execute the script by pressing <F5>. Enter various lengths of password.
   The next line to be executed will turn yellow.
4. Press <F11> to step through each line of the script and hover your mouse over the variables on each iteration.
5. When is the evaluation that controls the loop occurring?

After the script-lock has been executed at least once!

Task 15.1.4: Looping – ForEach

1. Open and execute the following script within the ISE script pane.
   C:\Pshell\Labs\Lab_15\Looping-ForEach.ps1
2. Highlight the code below, then press <F8> to execute.

```
Get-WinHomeLocation | Format-Table -autosize

$money = 12345678.99
$today = Get-Date
```

```
$cultureNames  = "ar-SA","da-DK","de-CH","de-DE","en-AU","en-GB","en-IN"
$cultureNames += "en-US","es-AR","es-ES","es-MX","fi-FI","fr-FR","ig-NG"
$cultureNames += "ja-JP","nn-NO","pt-BR","ru-RU","se-SE","th-TH","zh-HANS","en-NZ"
# note: use the ISE to ensure fonts are displayed correctly

ForEach ($cultureName in $cultureNames)
{
  $cultureInfo=New-Object Globalization.CultureInfo($cultureName)
  write-output "$cultureName : $($cultureInfo.Displayname)"
  $money.ToString("c",$cultureInfo) #currency
  $money.ToString("n",$cultureInfo) #numeric
  $today.ToString("D",$cultureInfo) #date
  $today.ToString("t",$cultureInfo) #time
  write-output "--------------"
}
```

3. Next, highlight the code below and press <F8> to execute.

```
ForEach ($disk in Get-CIMinstance Win32_LogicalDisk) {
  $drive=$disk.DeviceID
  $vol=$disk.VolumeName
  $type=[enum]::GetValues([System.IO.DriveType])[$disk.drivetype]
  $size=($disk.size / 1Gb)
  $free=($disk.FreeSpace / 1Gb)
  write-output "$drive '$vol' : $type : $($size.ToString('N2'))
($($free.ToSTring('N2')) `
  free)"
}
```

4. Note that we do not need to specify initialization or termination conditions. What terminates the loop?

Once we have processed the last element of the collection

Task 15.1.5: Looping – For

1. Open and execute the following script within the ISE script pane.
   C:\Pshell\Labs\Lab_15\Looping-For.ps1

```
# date is beginning of the year
$BoY=(Get-Date).AddDays( -(get-date).DayOfYear )

for ($i=1; $i -LT 365; $i++) {

  $day1=$BoY.AddDays($i);
  $day2=$BoY.AddDays($i+1)
#
# if one (and only one) has changed, then we flipped over!
#
  if ($day1.IsDaylightSavingTime() -XOR $day2.IsDaylightSavingTime()) {
    "Daylight Savings switch: {0:D}" -f $day2
  }
}
```

Discuss the init, condition test and post processing. We are looping no more than 365 days in this example. You may also flag the use of XOR to identify when the daylight savings change has occurred!

2. Highlight the For… line and press <F9>
3. Execute the script by pressing <F5>. The next line to be executed will turn yellow.
4. Press <F11> to step through each line of the script and hover your mouse over the $i variable on each iteration.
5. Note that the initialization of the variable $i, the test of the termination condition, and then the incrementing of $i, are similar to the requirements of the earlier "while" exercises, however it is now done in a single statement. For loops are best suited when the number of iterations is known.
6. Rather than pressing <F11> hundreds of times to finish the script, now press <F5> to run the script to completion.

Exercise 15.2: Branching

Introduction

Branching statements are language commands you can use to run lists of statements based on the results of one or more conditional tests.

Objectives

After completing this exercise, you will be able to:

- Understand the use and capabilities of the If and Switch statements

Task 15.2.1: Branching – If

1. Open the script from Module 5, C:\PShell\Labs\Lab_5\Create-BpaReport.ps1
2. We may not want to run Invoke-BPAModel every time the script runs. Adding the functionality to turn on or off sections within our script is easy. We can use a third parameter type, the switch parameter. As you may recall from Module 2, this is a Boolean, or true/false parameter.

```
param($serverName, $modelId, $filePath, [switch]$invokeModel)

if ($invokeModel)
{
    Invoke-Command -ScriptBlock {
    Invoke-BPAModel –ModelId $using:modelId
    } -ComputerName $serverName
}

Invoke-Command -ScriptBlock {
Get-BpaResult -ModelId $using:modelId
} -ComputerName $serverName |
ConvertTo-Html |
Out-File -FilePath “$filePath\DS-BPAReport.htm”
```

3. Execute the script using the command below, in the Windows PowerShell blue Console pane, with and without the –invokeModel switch parameter. If the switch parameter is used, the $invokeModel variable is created, the “if” statement returns true, and the section of code is executed.

```
C:\PShell\Labs\Lab_5\Create-BpaReport.ps1 –modelId
“Microsoft/Windows/DirectoryServices” `
-serverName “2012R2-DC” –filePath C:\PShell\Labs\Lab_5 –invokeModel
```

Task 15.2.2: Branching – If Else

1. Open and execute the following script within the ISE script pane.
   C:\Pshell\Labs\Lab_15\IF-ELSE-statement.ps1

```
$PROCESSORS   = $ENV:NUMBER_OF_PROCESSORS
$COMPUTERNAME = $ENV:COMPUTERNAME

if ($PROCESSORS -LT 4) {
  write-output "$COMPUTERNAME has $PROCESSORS CPU's."
  write-output "You will need to upgrade the number of processors."
}
elseif ($PROCESSORS -EQ 4){
  write-output "$COMPUTERNAME has $PROCESSORS CPU's."
  write-output "This is the minimum supported number of processors."
}
else {
  write-output "$COMPUTERNAME has $PROCESSORS CPU's."
  write-output "This is an adequate number of processors."
}

####
$logName = "Application"
$lastWeek = (get-date).AddDays(-7)
$lastweek -= $lastWeek.TimeOfDay
$LogErrors = get-eventlog -LogName $logName -EntryType Error -After $lastWeek

If ( $LogErrors.Count -GT 0) {
   If ($LogErrors.Count -EQ 1) {
     $errors = "Error" }
   else {
     $errors = "Errors"}

   Write-Output "$($LogErrors.Count) $logName $errors found since
$($lastWeek.ToString('D'))."
}
```

2. Highlight the if ($PROCESSORS -LT 4) … line and press <F9>
3. Execute the script by pressing <F5>. The next line to be executed will turn yellow.
4. Press <F11> to step through each line of the script and hover your mouse over the $PROCESSORS variable on each iteration.
5. Change the value of $PROCESSORS at line 1 to other numeric values and execute the script again to confirm the flow of logic.

Task 15.2.3: Branching – Switch (simple)

1. Open and execute the following script within the ISE script pane.
   C:\Pshell\Labs\Lab_15\Switch-Simple.ps1

```
switch ((Get-CIMinstance Win32_ComputerSystem).Model)
{
     "X70"       { $recipients = "Marketing"     }
     "S7"        { $recipients = "Marketing"     }
     "Aspire"    { $recipients = "Engineering"   }
```

```
    "Blade"     { $recipients = "Finance"       }
    "Ativ"      { $recipients = "Finance"       }
    "Inspiron"  { $recipients = "Engineering"   }
    "X240"      { $recipients = "Engineering"   }
    "IdeaPad"   { $recipients = "IT"            }
    "UX31A"     { $recipients = "IT"            }
    "Virtual Machine"  { $recipients = "IT"     }
    default     { $recipients = "Marketing"  }
  }
$recipients
```

2. Highlight the switch … line and press <F9>
3. Execute the script by pressing <F5>. The next line to be executed will turn yellow.
4. Press <F11> to step through each line of the script and note where the statement exits.
5. What value was shown for your virtual machine and why?

Consider the exact match, consider if no match and we fall through to default. Consider if some wildcard pattrn was used (and no –WildCard switch present).

Task 15.2.4: Branching – Switch (case sensitive)

1. Open and execute the following script within the Windows PowerShell ISE script pane. C:\Pshell\Labs\Lab_15\switch-CaseSensitive.ps1

```
$words = "MicrosOFT","MicroSoft","Microsoft","MiCroSoFt","MICROsoft"
$matched=0

switch ($words)
{
 "Microsoft" { write-host "'$_' matched!";$matched++}
 default     { write-host "do some default action"}
}
write-output "Switch with (default) no case sensitivity."
write-host "$matched matches."


$matched=0
switch -CaseSensitive ($words)
{
 "Microsoft" { write-host "'$_' matched!";$matched++}
 default     { write-host "call some default action"}
}
write-output "Switch with case sensitivity."
write-host "$matched matches."
```

2. Highlight the switch … line and press <F9>
3. Execute the script by pressing <F5>. The next line to be executed will turn yellow.
4. Press <F11> to step through each line of the script and note where the statement exits.
5. Consider the two results. Alter the values and test the statement again.

Task 15.2.5: Branching – Switch (wildcards)

1. Open (but do NOT execute) the following script within the Windows PowerShell ISE script pane. C:\Pshell\Labs\Lab_15\Switch-Wildcard.ps1

```
$filepath = "C:\PShell\Labs\Lab_15\OriginalFiles"
$archivePath = "$filepath\Archive\"

if (Test-Path $filepath)
{
  if (-NOT (Test-Path $archivepath))
  {
    New-Item -Path $filepath -Name "Archive" -ItemType Directory

    "2012","2013","KEEP" |
    ForEach-Object {New-Item -Path $archivepath -Name $_ -ItemType Directory}
  }

  switch -WildCard (Get-ChildItem "$filepath\*.log")
  {
    *2012*  { Move-Item   $_  -Destination $archivePath\2012 -Verbose -Force}
    *2013*  { Move-Item   $_  -Destination $archivePath\2013 -Verbose -Force}
    *KEEP*  { Move-Item   $_  -Destination $archivePath\KEEP -Verbose -Force}
    default { Remove-Item $_  -Verbose -Force}
  }

}
else {
 write-output "Not Found: $filepath"
}
```

Note the –Wildcard switch being used. Remind the student of the limitations of the wildcard set.

2. Highlight the switch … line and press <F9>
3. Execute the script by pressing <F5>. The next line executed will turn yellow.
4. Press <F11> to step through each line of the script and note where the statement exits.
5. Confirm that the archive folder contains the files matching the wildcard patterns.
6. After stepping through a few iterations with <F11>, press <F5> to run the script to completion.

Task 15.2.6: Branching – Switch (Regular Expression)

1. Open and execute the following script within the Windows PowerShell ISE script pane. C:\Pshell\Labs\Lab_15\ Switch-RegEx.ps1

```
$array = "123.456","abcdefg","THX-138","Rhubarb", "0414 987 058","MicroSoft"

switch -RegEx ($array)
{
 "^\d{1,3}\."        {write-host "'$_' begins with a numeric then a dot"}
 "\d{3}$"            {write-host "'$_' ends with three digits"}
 "[A-Z]-\d+"         {write-host "'$_' contains ALPHA-NUMERIC dash numbers"}
 "\d+\s\d{3}\s\d{3}" {write-host "'$_' looks like a phone number"}
 default             {write-host "'$_' had no match..."}
}
```

Note the –RegEx switch being used! FULL power of the regular expression engine is available.

```
# NOTE: more advanced, add in something for luke@contoso.com.au
#"\w+\@\w+[\.\w+]+"  "looks like an email address"
```

2. Highlight the switch … line and press <F9>
3. Execute the script by pressing <F5>. The next line to be executed will turn yellow.
4. Press <F11> to step through each line of the script and note where the statement exits.
5. Create alternate values in the array and test if the regular expression patterns match.
   See: Get-Help about_regular_expressions

Task 15.2.7: Branching – Switch (input from filename)

1. Ensure you have launched ISE with "Run as Administrator".
2. Open and execute the following script within the Windows PowerShell ISE script pane. C:\Pshell\Labs\Lab_15\ Switch-Filename.ps1

```
#Requires -RunAsAdministrator

  copy-item $ENV:WinDir\WindowsUpdate.Log $ENV:TEMP\WindowsUpdate.Log -Force
  $errcount=$warncount=0

  switch -wildcard -CaseSensitive -File "$ENV:TEMP\WindowsUpdate.Log"
  {
    "*ERROR*"   {write-host "Panic! Panic!";$errcount++}
    "*WARNING*" {write-host "Worry! Worry!";$warncount++}
  }
  write-output "Errors:   $errcount"
  write-output "Warnings: $warncount"
```

We use a copy in TEMP because Switch –FILE tries to open the source as read/write and will FAIL on the original in C:\Windows

3. Execute the script by pressing <F5>.
4. Explore the WindowsUpdate.log file (it can be found under c:\windows) and alter the patterns to match things you wish to highlight.

Task 15.2.8: Branching – Switch (and the break statement)

1. Ensure you have launched ISE with "Run as Administrator".
2. Open and execute the following script within the Windows PowerShell ISE script pane. C:\Pshell\Labs\Lab_15\ Switch-Break.ps1

```
#Requires -RunAsAdministrator

# switch can iterate an array/collection/pipeline AND be told to halt.

 $SinceWhen = "TODAY","YESTERDAY","LOGIN","BOOT"

 Switch ($SinceWhen | Sort-Object) {
   "BOOT"      {
                 $OS = Get-CIMinstance Win32_OperatingSystem
                 $sinceTime = $OS.LastBootUpTime
                 Write-Output "Boot $sinceTime"
               }
```

```
   "LOGIN"     {
                 $loginSession = Get-CIMinstance -class Win32_LogonSession
                 $sinceTime = $loginSession.StartTime
                 Write-Output "Login $SinceTime"
               }
   "TODAY"     {
                 $sinceTime = (Get-Date)-(Get-Date).timeofday
                 Write-Output "Today $sinceTime"
               }
   "YESTERDAY" {
                 $sinceTime = ((Get-Date)-(Get-Date).timeofday).AddDays(-1)
                 Write-Output "Yesterday $sinceTime"
               }
 }
# what happens if I add a ";Break" to one of the matches?
```

3. Highlight the switch … line and press <F9>
4. Execute the script by pressing <F5>. The next line to be executed will turn yellow.
5. Press <F11> to step through each line of the script and note where the statement exits.
6. Add a line to one of the script blocks after a Write-Output with the command “break” and re-run.
7. Note the change in behavior!

Exercise 15.3: Break and Continue keywords

Introduction

Break and Continue keywords are used to alter the default flow within a flow control statement.

Objectives

After completing this exercise, you will be able to:

- Understand the use and capabilities of Break and Continue.

Task 15.3.1: Flow Control – Break/Continue

Flow control statements can immediately exit, or continue, a loop or script block's execution.

1. Open and execute the following script within the ISE script pane. C:\Pshell\Labs\Lab_15\ BreakContinue.ps1

```
$i=0
foreach ($x in 1..100) {
  $i += $x;
  if ($i % 5) { "Jump to end"; continue}
  $i
  if ($i -gt 1770) {"I'm Out!"; break}
}

write-output "Note: Continue jumps to the bottom of the loop."
write-output "Note: Break exits the loop completely and the next"
write-output "      section of the script will then execute."
```

2. Alter the numeric values in the if statements and note how the execution flows.

Task 15.3.2: Flow Control – Exit

A Windows PowerShell script exits after the last line of code executes. Within functions, it is possible to "Return" early from the current scope. This behaves like a break statement, but for a function. The same ability exists within Scripts. The Exit statement can force the entire script to exit early, regardless of the current execution location.

1. Open and execute the following script within the Windows PowerShell ISE script pane. C:\Pshell\Labs\Lab_15\ExitScript.ps1

```
$dayToIgnore = "Monday"

function Get-PublicSMBInfo() {
 if ( ((get-date).DayOfWeek) -EQ $dayToIgnore ) {
   "I don't work on $dayToIgnore!"
   EXIT
 }
 return ( Get-SMBshare | where {$_.Description -Match "^Public"} )
}

Get-PublicSMBinfo | Format-Table

Write-Host "Will never get here on '$dayToIgnore'!"
```

2. Change the value of $dayToIgnore variable and re-run. Note the result when EXIT occurs.