Advanced Scripting   
Comparison, Logical, and Other Operators

Last Updated: 1/2/2024 4:12 PM Version 1  
Document Prepared for: CYBER360 Student

# Name Click here to enter name ID Click here to enter id

# Instructions

Answer all questions directly in this document. You will save and upload this completed document as your homework submission.

# Overview

This exercise presents a few practice drills involving PowerShell operators.

# Setup

## Requirements

* PowerShell

# Task 1—Ordinal Comparison Operators

The “ordinal” or “lattice” comparison operators are:

|  |  |  |  |
| --- | --- | --- | --- |
| Operator (Default) | Case  Sensitive version | Case  Insensitive version | Meaning |
| -eq | -ceq | -ieq | Equal |
| -ne | -cne | -ine | Not equal |
| -gt | -cgt | -igt | Greater than |
| -ge | -cge | -ige | Greater than or equal to |
| -lt | -clt | -ilt | Less than |
| -le | -cle | -ile | Less than or equal to |

## Steps

1. For each of the expressions predict what the value will be ($true or $false). Then type the expression in PowerShell. Record the result. Then describe the result, and if it’s different than what you predicted, explain *how* the operator behaved differently than your intuition expected.

|  |  |  |  |
| --- | --- | --- | --- |
| Expression | Prediction | Result | Description |
| '' -eq $null | Click or tap here to enter text. | Click or tap here to enter text. | Click or tap here to enter text. |
| 'red' -eq 'red' | Click or tap here to enter text. | Click or tap here to enter text. | Click or tap here to enter text. |
| 'red' -eq 'Red' | Click or tap here to enter text. | Click or tap here to enter text. | Click or tap here to enter text. |
| 'red' -ceq 'Red' | Click or tap here to enter text. | Click or tap here to enter text. | Click or tap here to enter text. |
| 1 -gt 1 | Click or tap here to enter text. | Click or tap here to enter text. | Click or tap here to enter text. |
| 12 -gt '2' | Click or tap here to enter text. | Click or tap here to enter text. | Click or tap here to enter text. |
| '12' -gt 2 | Click or tap here to enter text. | Click or tap here to enter text. | Click or tap here to enter text. |

1. When the left-hand operator is a collection (e.g. array), each item is compared to the right-hand operand, and if the comparison is true the item is passed along the pipeline.
   1. Try the following  
      1,2,3,4 -ge 2
      1. What is returned?Click or tap here to enter text.
   2. Try  
      'a','b','c','a' -lt 'c'
      1. What is returned? Click or tap here to enter text.

# Task 2—-Like and -NotLike, Basic Pattern Matching

-like and -notlike use simple wildcard pattern matching. The wildcard characters are:

|  |  |  |
| --- | --- | --- |
| **Wildcard** | **Matches** | **Examples** |
| \* | Match any sequence of characters | \*.log |
| ? | Match any single character | 1225????.log |
| [<char>-<char>] | Match one of a range of characters | SiO2[1-6][a-b] |
| [<char><char>] | Match one of a specified set characters | [02468][ace]xyz |

## Steps

1. Many cmdlets allow basic wildcard matching for parameters. **Get-Command** is a good example.
   1. To get a list of all the commands that start with New use the argument **New-\***  
      get-command New-\*
   2. To get all commands that contain the word *process*, wrap **process** between **\*** splat wildcards:  
      get-command \*process\*
   3. Get all the commands that start with *Get-* and the noun starts with the letters a through p:  
      Get-Command Get-[a-p]\*
2. The **-like** and **-notlike** operators can be used wherever you need a comparison. Given the following, predict, then verify and record the expression result with PowerShell

|  |  |  |
| --- | --- | --- |
| Expression | Prediction | Result |
| **'red' -like 'r\*'** | Click or tap here to enter text. | Click or tap here to enter text. |
| **'red' -like 'R\*'** | Click or tap here to enter text. | Click or tap here to enter text. |
| **'red' -clike 'R\*'** | Click or tap here to enter text. | Click or tap here to enter text. |
| **'red' -notlike 'red'** | Click or tap here to enter text. | Click or tap here to enter text. |
| **8675309 -like '[5-9]??53\*'** | Click or tap here to enter text. | Click or tap here to enter text. |

# Task 3— -Match and -NotMatch, Regular Expression Matching

The **-match** operator uses regular expression (“regex”) matching. Regular expressions are a more complex pattern matching system than the wildcards used by **-like** and **-notlike**. To learn about regular expressions there are numerous tutorials and references online.

## Steps

1. The simplest and probably the most used regular expressions determine whether a string contains a specific sequence of characters. Get familiar with it:

|  |  |  |
| --- | --- | --- |
| Expression | Prediction | Result |
| **'99 red balloons' -match 'Red'** | Click or tap here to enter text. | Click or tap here to enter text. |
| **'99 red balloons' -notmatch 'Red'** | Click or tap here to enter text. | Click or tap here to enter text. |
| **'99 red balloons' -cmatch 'Red'** | Click or tap here to enter text. | Click or tap here to enter text. |

1. A regex “or” operation has alternative strings between parentheses and separated by | vertical bars. Use regex “or” alternatives to match one among a list words:

|  |  |  |
| --- | --- | --- |
| Expression | Prediction | Result |
| **'99 red balloons' -match '(red|blue|green)'** | Click or tap here to enter text. | Click or tap here to enter text. |
| **'99 orange balloons' -match '(red|blue|green)'** | Click or tap here to enter text. | Click or tap here to enter text. |
| **'99 blue balloons' -cmatch '(red|blue|green)'** | Click or tap here to enter text. | Click or tap here to enter text. |

1. Square brackets [] can be used to specify ranges or sets of single characters to be matched. These patterns are identical to the corresponding character wildcards of the **-like** operator:

|  |  |  |
| --- | --- | --- |
| Expression | Prediction | Result |
| **'99 blue balloons' -match '[0-9]'** | Click or tap here to enter text. | Click or tap here to enter text. |
| **'99 blue balloons' -match '[0-9] [0-9]'** | Click or tap here to enter text. | Click or tap here to enter text. |
| **'99 blue balloons' -match '[0-9] [0-9] [0-9]'** | Click or tap here to enter text. | Click or tap here to enter text. |

1. Special regex *anchors* ^ and $ don’t match a character, but a *position* instead: ^ matches the beginning of a string, $ matches the end of a string:

|  |  |  |
| --- | --- | --- |
| Expression | Prediction | Result |
| **'99 blue balloons' -match '^[0-9]'** | Click or tap here to enter text. | Click or tap here to enter text. |
| **'99 blue balloons' -match '^blue'** | Click or tap here to enter text. | Click or tap here to enter text. |
| **'99 blue balloons' -match 'balloon$'** | Click or tap here to enter text. | Click or tap here to enter text. |
| **'99 blue balloons' -match 'loons$'** | Click or tap here to enter text. | Click or tap here to enter text. |

1. Regex wildcards:
   1. The period **.** is the regex wildcard that matches any single character (analogous to the question mark **?** wildcard of the **-like** operator).
   2. The two-character sequence “dot-splat” **.\*** is the regex wildcard that matches any sequence of any characters (analogous to the splat **\*** wildcard of the **-like** operator).
   3. Any other character, followed by a splat **\***, is a regex wildcard that matches any repeated sequence of that character only. Example: the regex pattern **'=\*'** matches any of **''**, **'='**, **'=='**, **'==='**, etc.

|  |  |  |
| --- | --- | --- |
| Expression | Prediction | Result |
| **'99 red balloons' -match '99 r.d b......s'** | Click or tap here to enter text. | Click or tap here to enter text. |
| **'99 blue balloons' -match 'blue.\*loon'** | Click or tap here to enter text. | Click or tap here to enter text. |
| **'99 red balloons' -match '^9\*'** | Click or tap here to enter text. | Click or tap here to enter text. |
| **'99 red balloons' -match '99 red [a-z]\*'** | Click or tap here to enter text. | Click or tap here to enter text. |

1. To match a specific character, like dot **.**, splat **\***, hat **^**, or dollar **$**, it must be “escaped” by prefixing with a backslash (**\.** or **\\*** or **\^** or **\$**) so that they are not misinterpreted as wildcards.

|  |  |  |
| --- | --- | --- |
| Expression | Prediction | Result |
| **'Price\* $9.90' -match '\$9\.90$'** | Click or tap here to enter text. | Click or tap here to enter text. |
| **'Price\* $9.90' -match '^Price\\* '** | Click or tap here to enter text. | Click or tap here to enter text. |

# Task 4—Logical Operators

Fill in the following logical truth tables

## Steps

1. For the -and operator to return True both operands must be True.

|  |  |  |
| --- | --- | --- |
| -and | $true | $false |
| $true | Click or tap here to enter text. | Click or tap here to enter text. |
| $false | Click or tap here to enter text. | Click or tap here to enter text. |

1. For the -or operator, at least one of the operands must be True. (Think “left or right or both.”)

|  |  |  |
| --- | --- | --- |
| -or | $true | $false |
| $true | Click or tap here to enter text. | Click or tap here to enter text. |
| $false | Click or tap here to enter text. | Click or tap here to enter text. |

1. For the -xor operator, one operand must be True and the other operand must be False. (“left or right but not both.”)

|  |  |  |
| --- | --- | --- |
| -xor | $true | $false |
| $true | Click or tap here to enter text. | Click or tap here to enter text. |
| $false | Click or tap here to enter text. | Click or tap here to enter text. |

1. The -not operator is a unary operator. It returns the inverse of the operand.

|  |  |
| --- | --- |
|  | -not |
| $true | Click or tap here to enter text. |
| $false | Click or tap here to enter text. |

# Task 5— -Split

The **-split** operator breaks a string into an array of strings at the matched delimiter.

## Steps

1. Split on a space  
   'I love cilantro' -split ' '
   1. What was returned? Click or tap here to enter text.
2. Split on the letter i  
   'I love cilantro' -split 'i'
   1. What was returned? Click or tap here to enter text.
3. Use more than one character as a delimiting phrase  
   'a flea on a fly on a wart on a frog on a knot on a log' -split ' on a '
   1. What was returned? Click or tap here to enter text.
4. Use a square-bracketed character set (regular expression wildcard) to specify delimiters:  
   'query=src&start=0+end=4+limit=9' -split '[&+]'
   1. What was returned? Click or tap here to enter text.
5. Use **-split** as a unary operator to split on white space:  
   -split 'I am really bad at spacing !'

# Task 6—-Join, .. (range) Operators

The **-join** operator combines an array of items into a single string. A string operand to the right of **-join** provides characters concatenated between the array elements. The **..** (two periods) operator is a shortcut that creates an array of integers. The operands specify the bounds of the sequence. *(If you’re using PowerShell Core, you can also use it to create an array from a range of characters.)*

## Steps

1. Join an array with a comma delimeter  
   1,2,3,4 -join ','
2. Use the range operator to create a much larger array, then join it up with a space delimeter  
   12..49 -join ' '
3. Use **-join** as a unary operator to mash everything together with no separating delimiters.  
   -join (4..0)
   1. What was returned? Click or tap here to enter text.

# Task 7—Property and Method Operators

## Steps

1. The . operator is used to access an object’s instance properties and methods. (This is sometimes described as *passing a message* to an object.) Methods always require parentheses () after the method name.
   1. Put an object into a variable  
      $gem='Diamond'
   2. Access a property  
      $gem.Length
      1. How many characters are in $gem? Click or tap here to enter text.
   3. Call a method with no parameters  
      $gem.ToUpper()
      1. What was returned? Click or tap here to enter text.
   4. Call a method with a parameter  
      $gem.Substring(3)
      1. What was returned? Click or tap here to enter text.
   5. When calling a method with multiple parameters use a **,** (comma) to separate parameter values. Call a method with multiple parameters  
      $gem.Substring(3,2)
      1. What was returned? Click or tap here to enter text.
2. The **::** operator is used to access *static* properties and methods of a class. Static members are used *without* an instance of an object.
   1. Call a static method of the string type  
      [string]::IsNullOrEmpty($gem)
      1. What was returned? Click or tap here to enter text.
   2. Access a static property  
      [datetime]::now
      1. What was returned? Click or tap here to enter text.

# Task 8—Grouping and Subexpressions

## Steps

1. Parenthesis are used to group expressions, or force order of operations.
2. Group expression. This example forces a multiply operation to complete *before* a method call:  
   (1 \* 2.5).gettype()
   1. What is returned? Click or tap here to enter text.
3. Group the results of a cmdlet  
   (Get-Process).count
   1. What is returned? Click or tap here to enter text.
4. **$()** groups/interpolates statements. (May contain loops, statements separated by semicolon **;** )  
   "Hi $($n=read-host 'What is your name?';$n.toupper())"
   1. What is returned? Click or tap here to enter text.
5. **@()** is the same as **$()** except it *always* returns an array.  
   $a=$($n=read-host 'What is your name?';$n.toupper())  
   $b=@($n=read-host 'What is your name?';$n.toupper())
   1. What type is the value of **$a**? Click or tap here to enter text.
   2. What type is the value of **$b**? Click or tap here to enter text.

# Task 9—Redirection operators

The redirection operator “redirects” the output of a command to a file. The **>** operator creates a new file whereas the **>>** operator appends to an existing file (if the file does not exist, it creates the file).

## Steps

1. Make your system temporary folder your current directory: **cd \TEMP** (Windows), **cd /tmp** (Linux)
2. Redirect standard output to a file
   1. Save you process list to a file  
      get-process > profile.txt
   2. View the contents of the file  
      Get-Content profile.txt
   3. Add a list of aliases to the end of **profile.txt**  
      get-alias >> profile.txt
   4. Confirm the data was added to the file.
3. Redirect error output. PowerShell has several output streams. Stream #2 is the error stream.
   1. Create some folders to work with  
      1..4|%{New-Item -Type Directory -Name "out$\_"}
      1. Use Get-ChildItem to see the new folders.
      2. You should see four of them: out1, out2, out3, out4
   2. Delete the files with some errors  
      0..5|%{Remove-Item "out$\_"}
   3. This should have deleted the files you created and tried to delete two files that do not exist, causing a couple of errors.
   4. Recreate the files. *(Reminder: don’t retype it. Use the up-arrow key to get it from your history)*  
      1..4|%{New-Item -Type Directory -Name "out$\_"}
   5. Now delete the files again, this time redirecting the errors to a file named errors.txt  
      0..5|%{Remove-Item "out$\_"} 2> errors.txt

Describe what is in errors.txt? Click or tap here to enter text.

# Deliverable

Upload this document with completed answers to I-Learn Canvas.