ScriptingAnswers.com

Windows PowerShell Intermediate

LAB GUIDE

Lab 1-1
Advanced Filtering and Cmdlet Techniques

Bear in mind that most of these tasks will require you to learn something new about one or more cmdlets – don't forget to use the **help** functionality!

Task #1

Here's your task:
Get a listing of all DLL files in your System32 or System64 folder. Use –include, -exclude, or –filter rather than specifying a wildcard in the path.

Here's a tip:
The **Dir** command will do the job. Ask for **Help** on using the filtering parameters.

Task #2

Here's your task:
Use WMI to get a listing of the free space on all local hard drives. Use no more than two cmdlets on the pipeline, and do not use the –query parameter. Display both the drive letter and the free space.

Here's a tip:
You'll need to find the filtering parameter for the WMI cmdlet. Remember that its comparison syntax will be different from what PowerShell itself uses.

Here's a tip:
Your second pipeline cmdlet should *select* the property that you want to display from the objects in the pipeline.

Task #3

Here's your task:
Create an HTML file that shows all running processes. Make the title of the HTML page "Running Processes" (so that the title appears in the browser's window title bar). Put the text "Running Processes" above the HTML table in the HTML file.

Here's a tip:
Use **Help** to explore the capabilities of the ConvertTo-HTML cmdlet

Task #4

Here's your task:
Display a list of unique process names (e.g., "svchost" is usually running several instances, but would be displayed only once).

Here's a tip:
There's a cmdlet that can *select* the properties you want from the objects in the pipeline; see if this cmdlet has any additional capabilities.

Task #5

Here's your task:
Write a one-liner that displays the amount of free space, in gigabytges (GB), on each local hard disk. Write the one-liner so that the phrase "FREE SPACE REPORT" appears as the first line of output, then the free space, then the phrase "COMPLETE" as the last line.

Here's a tip:
You can't use **Select-Object** for this. You'll need to work with each object one at a time in the pipeline in order to divide the free space from bytes to gigabytes. The same cmdlet that lets you work with one object at a time will also allow you to output something before and after it begins working with objects.

Lab 2-1
Leveraging Objects

Before beginning, use Notepad to create a text file with several lines of sample text. Ensure that some lines of text start with spaces. You'll work with this text file in several of the tasks for this lab.

Task #1

Here's your task:
Using your text file, display each line with any leading or trailing spaces removed. Write this as a one-liner.

Here's a tip:
Remember to pipe a string to **Get-Member** to see that the object can do.

Task #2

Here's your task:
Using your text file, display each line in uppercase. When displaying each line, replace any occurrences of the letter "X" with the letter "Z." Write-this as a one-liner.

Here's a tip:
Remember to pipe a string to **Get-Member** to see that the object can do.

Here's a tip:
Methods of a String object return strings themselves. Those new strings have all the capabilities of any String object. So, something like this is valid:

```
"Hello".ToLower().StartsWith("H")
```

Task #3

Here's your task:
Using your text file, display the position of the letter "X" within each line. Display -1 for lines which do not contain the letter "X."

Here's a tip:
Remember – **Get-Member**!

Task #4

Here's your task:
Display the date from one year ago, taking leap years into account.

Here's a tip:
Send a date to **Get-Member** to see what it can do.

Task #5

Here's your task:
Take the date from four years ago and display it as a file system date/time value.

Here's a tip:
Send a date to **Get-Member** to see what it can do.

Lab 3-1
Error Trapping and Handling

Task #1

Here's your task:
Write a function that uses WMI to retrieve the user account that a specified service runs under. The function should accept the service name and a computer name as input parameters; it should return the login name that the service uses, or the string "not reachable" if the computer cannot be contacted.

Here's a tip:
Review the information from module 3 for more information.

Lab 4-1
Debugging Techniques

Task #1

Here's your task:
You'll find the starting point for this script in Lab4-1.ps1, which is included in your Lab Guide folder.

Here's a tip:
Remember to use the tools and techniques that were discussed:

- Develop an expectation of what the script should do
- Get "out of the script" to test things like WMI queries
- Use Write-Debug and $DebugPreference to write trace code
- Use the shell's interactive debugger to step through the code and examine variables as the script runs

```
# Should display OS build number and SP version for specified computer
New-Variable $FLAG_BUILD 0 -option constant
New-Variable $FLAG_SPVER 1 -option constant

Function Ping-Computer ($computre) {

	$wmi = gwmi -query "SELECT * FROM Win32_PingStatus WHERE Name = '$computer'"

	if ($wmi.statuscode) {
		return $true
	} else {
		return $false
	}
}

function Get-OSInfo ($computer,$flag) {

	$wmi = gwmi win32_operatingsystem -comp $computer
	switch ($flag) {
		0 { $result.buildnumber; break }
		1 { $result.servicepackmajorversion; break }
	}
}

$computer = "localhost"
if (Ping-Computer $computer) {

	Get-OSInfo $computer,$FLAG_BUILD
	Get-OSInfo $computer,$FLAG_SPVER
}
```

Lab 5-1
Regular Expressions

For each of these tasks you will be asked to write a regular expression that accepts ("True" match) a particular type of data. All noncompliant data should result in a non-match ("False"). Use the appropriate regex operator to ensure the appropriate True or False match with your regex.

Task #1

Here's your task:
Write a regex that matches on a UNC, in the format \\Server2\Share. Longer UNCs such as \\Server2\Share\Folder\File should also be accepted.

Here's a tip:
Remember to use anchors and character classes appropriately, and remember that special characters are escaped with a backslash.

Task #2

Here's your task:
Write a regex that accepts all e-mail addresses in the format firstname.lastname@company.com; the "company.com" portion is static—each e-mail address must end in the literal string "company.com."

Here's a tip:
Remember to use anchors and escape characters appropriately.

Task #3

Here's your task:
Write a regex that accepts user names in the format "Last, First".

Here's a tip:
Remember to use character classes to detect whitespace.

Task #4

Here's your task:
Write a regex that accepts a date, in the format MM/DD/YYYY or DD/MM/YYYY, ensuring a two-digit month, two-digit day, and four-digit year. Use slashes only, not dashes.

Here's a tip:
Remember that you can specify minimum/maximum match counts in curly braces.

Task #5

Here's your task:
Write a regex that will reject any string containing a single or double quotation mark.

Here's a tip:
Remember to test your regex against data which *should* be rejected, to make sure it is being rejected properly. This regex must return True if a quotation mark is in the string!

Here's a tip:
To include a double quotation mark inside a PowerShell string, type the quote twice.

Lab 6-1
Advanced Modularization

Task #1

Here's your task:
Write a function named Get-DriveInfo. It must be a filtering function, accepting computer names (strings) from the pipeline. For each computer name piped in, use WMI to retrieve the Win32_LogicalDisk class *only* for local hard disks on the specified computer.

The function must output the following properties:

- Computer name
- Drive letter
- Free space

Write a one-liner which utilizes your function to display a list of computer names and drive letters, including the free space on each drive.

Lab 6-2
Advanced Modularization

Task #1

Here's your task:
Write a function named Get-DriveInfo. It must be a filtering function, accepting computer names (strings) from the pipeline. For each computer name piped in, use WMI to retrieve the Win32_LogicalDisk class *only* for local hard disks on the specified computer.

The function must output a custom object with the following properties:

- Computer name
- Drive letter
- Free space (in megabytes, with no fractional portion)

Write a one-liner which utilizes your function to display a list of computer names and drive letters of drives with less than 10MB of free space, with the drives having the least free space listed first.

Here's a tip:
To convert a fraction number (stored in $f, for example) to a whole number:

```
$i = $f –as [int]
```

Lab 7-1
Importing and Exporting Objects

Task #1

Here's your task:
Write a one-liner that can be scheduled to run on a server, which will export its started services into an XML file.

Write a second one-liner which imports the XML file and generates a list of service names which were started earlier, but not started now, or vice-versa.

Here's a tip:
Pipe process to Get-Member to see their properties. Is there a property that tells you if the process is responding to Windows?

Task #2

Here's your task:
Create a CSV file of all services configured to start automatically. For each service, include only its name, current status, and its logon name. Do this as a one-liner.

Here's a tip:
You can't use Get-Service for this – you'll have to get a list of services using another technique.

Here's a tip:
The Win32_Service WMI class has a StartName and a StartMode property which will be useful.

SOLUTIONS

Lab 1-1 Solutions

Task #1

Here's the solution:
Run this command:

```
Dir C:\Windows\System32 –filter "*.dll"
```

Task #2

Here's the solution:
Run this command:

```
gwmi win32_logicaldisk -filter "drivetype=3" | `
 select deviceid,freespace
```

Task #3

Here's the solution:
Run this command:

```
ps | convertto-html -title "RUNNING PROCESSES" -body `
 "Running Processes" | out-file c:\test\test.html
```

Task #4

Here's the solution:
Run this command:

```
ps | select name -unique
```

Task #5

Here's the solution:
Run this command:

```
gwmi win32_logicaldisk -filter "drivetype=3" | `
 % -begin { "FREE SPACE REPORT" } -end `
 { "COMPLETE" } -process { $_.Freespace / 1GB }
```

Lab 2-1 Solutions

Task #1

Here's the solution:
Run this command:

```
Gc c:\test\text.txt | % { $_.Trim() }
```

Task #2

Here's the solution:
Run this command:

```
Gc c:\test\text.txt | % { $_.ToUpper()Replace("X","Z") }
```

Task #3

Here's the solution:
Run this command:

```
Gc c:\test\text.txt | % { $._.IndexOf("X") }
```

Task #4

Here's the solution:
Run these commands:

```
$d = Get-Date
$d.AddYears(-1)
```

Task #5

Here's the solution:
Run these commands:

```
$d = Get-Date
$d.AddYears(-4).ToFileTime()
```

Lab 3-1 Solutions

Task #1

Here's the solution:

```
Function Get-ServiceAccount {
        param (
                [string]$computer = "localhost",
                [string]$service = "WebClient"
        )
        $ErrorActionPreference = "SilentlyContinue"
        trap {
                return "not reachable"
        }
        $services = gwmi win32_service -comp $computer -filter "name='$service'" -ea stop
        foreach ($service in $services) {
                return $service.startname
        }
}

Get-ServiceAccount "server2" "WebClient"
```

Lab 4-1 Solutions

Task #1

Here's the solution:

```
# variable names don't get $ in variable cmdlets
New-Variable FLAG_BUILD 0 -option constant
New-Variable FLAG_SPVER 1 -option constant
# Typo in argument name
Function Ping-Computer ($computer) {
        # correct property is Address, not name
        $wmi = gwmi -query "SELECT * FROM Win32_PingStatus WHERE Address = '$computer'"
        # statuscode is 0 for success, which PSH interprets as False
        if ($wmi.statuscode -eq 0) {
                return $true
        } else {
                return $false
        }
}

function Get-OSInfo ($computer,$flag) {
        # $wmi is a collection - must enumerate through it
        # can't access properties directly from $wmi
        # PingResult (above) is an exception to this rule in WMI
        $wmi = gwmi win32_operatingsystem -comp $computer
        foreach ($result in $wmi) {
                switch ($flag) {
                        0 { $result.buildnumber; break }
                        1 { $result.servicepackmajorversion; break }
                }
        }
}

$computer = "localhost"
if (Ping-Computer $computer) {
        # don't use commas between parameters
        Get-OSInfo $computer $FLAG_BUILD
        Get-OSInfo $computer $FLAG_SPVER
}
```

Lab 5-1 Solutions

Task #1

Here's the solution:

```
^\\\\\w*\\\w*
```

Task #2

Here's the solution:

```
^\w*\.\w*@company\.com$
```

Task #3

Here's the solution:

```
^\w*, \w*$
```

Task #4

Here's the solution:

```
^\d{2}/\d{2}/\d{4}$
```

Task #5

Here's the solution:

```
String –notmatch "[""']"
```

Lab 6-1 Solutions

Task #1

Here's the solution:

```
function Get-DriveInfo {
        PROCESS {
                $drives = gwmi win32_logicaldisk -comp $_ -filt "drivetype=3"
                foreach ($drive in $drives) {
                        $obj = New-Object psobject
                        $obj | Add-Member noteproperty computername $_
                        $fs = $drive.freespace / 1mb
                        $fs = $fs -as [int]
                        $obj | Add-Member noteproperty freespace $fs
                        $obj | Add-Member noteproperty driveletter $drive.deviceid
                        Write-Output $obj
                }
        }
}

@("localhost") | Get-DriveInfo | sort freespace -desc | `
 where { $_.Freespace -lt 10 } | select computername,driveletter
```

Lab 7-1 Solutions

Task #1

Here's the solution:
Run this command:

```
Gsv | export-clixml c:\test\services.xml
```

Then, run:

```
$s = import-clixml c:\test\services.xml
Diff $s (gsv) –prop displayname
```

Task #2

Here's the solution:
Run this command:

```
Gwmi win32_service | where { $_.StartMode –eq "Auto" } | `
 Select Name,StartName,State | Export-CSV c:\test\autostart.csv
```