Powershell basics

Understand the following:

Basic movement commands

Getting help

Aliases

Variables

Environment variables

Piping

$_ and $i concept

Get-member

Multithreading/multitasking with jobs

Parameters

Internal and external commands

Example of an external command is attrib, ping, etc

Example of internal command is type, dir etc…

CMD.exe is the PPID of all internal commands

Powershell = cmdlets (are verb-action)

Example get-process, get-childitem, get-executionpolicy

Data structures

Determines how you can interact with the information returned to the screen

External commands=string

Powershell internal commands= objects

Objects have properties and methods

Properties=data

Methods= functions that you can run on those properties

Pipe cmdlets into GM (get-member) to view the methods and properties

Addressing properties and methods of an object

When you are addressing a propery or method the syntax is:

Object.<property name>

Or

Object.<method>

$obj= tasklist

$obj.length

$obj.split( ) Note: Whatever you put into the parenthesis is the character you intend to split on.

Varaibles

$obj= tasklist

If you want to get fancy and do a longer command, you must use command expansion, piping, ifs, fors, etc.....

Ex: $A=$(get-something | where something……..)

Viewing other properties that aren't output to the screen by default

Get-process | gm

Get process | select threads, processname, id

Outputting to a file

$proc=$(get-process)

echo $proc >>proc.txt

notepad.exe proc.txt

Reading in the contents of a text file

Get-contents or gc

Ex: $a= gc file.txt

Arithmetic

$A=1

$B=10

$C="hello"

$D= $($A+$B; $C)

Iteration

$proc= get-process

$proc | % {$_.modules}

Putting multiple things in one script block

$proc | % {$A=$($_.id); $B=$($_.processname); echo “$B--$A”}

For each individual entry in $proc it will create $A and $B, then echo them. It will finish this then do the same for the next entry

Note that it is very important to understand the data-structure of what you are piping in your script blocks, remember you cannot $_.id on a string. Your line will fail to execute.

Starting and stopping services

Get-service

Get-service | where {$_.name -eq “wudfsvc”}

Our arch-nemesis WUDFSVC has been killed with 2 methods

- Method 1: find the pid of the service with sc querex and do a taskkill /F /IM <PID>
- Method 2: wmic service where name=”wudfsvc” call stopservice

Get-service | where {$_.status -eq “Stopped”}

$svc= $(get-service | where {$_.name -eq <something>}

WMI OBJECTS

Get-wmiobject or gwmi for short

ex: gwmi win32_process

gwmi win32_process | select-object processname

Foreach

This can be referenced with %

It iterates over each item in a list

Ex: get-service | % {echo “$_ is a service”}

Indexing

$A=$(Get-process)

$A[0] (or 1 or 2 or 3 and so on…)

Remember that indexes always start at 0

FOR loops

General structure of a FOR statement can be thought of as: FOR;WHILE;DO

For ($i=0; $i -lt 20; $i++)

FOR: i equals 0 initially

WHILE: i is less than 20

DO: increment i by 1

You can also use carriage returns in lieu of semicolons

You can make the value being compared against into a variable

Ex:

```
$value= 20
For ($i=0; $i -lt $value; $i++)
{echo $i}
```

Where

Where object can be referenced as ?

It runs a test construct where if TRUE, the object is passed through the pipe.

```
Get-process | ? {$_.id -gt 1000}
```

Like is another useful tool as well.

```
Get-process | ? {$_.id -like “*1*”}
```

While Loops

Example 1:

```
$i=0
While ($i -lt 20)
{
Echo “hello”
$i++
}
```

Example 2: Infinite loop

```
While ($true)
{
Echo “hello”
}
```

Example 3: Breaking out of a loop

```
While ($i -ne “<currenttime plus one minute>”)
```

```
{
$i=$(get-date -format t)
Echo “hello”
Sleep 2
}
```

IF

```
$file= Get-content file1.txt
$filelength= $file.length
        If ($filelength -gt 500)
                {del $file)
        If ($filelength -lt 500)
                {break}
```

Functions

Kind of like making a variable except it contains a block of code (or a whole script for that matter)

EX:

```
Function enumerate-network
        {get-nettcpconnection |where {$_.state -eq “Listening”}
        Arp -a
        Ipconfig /all}
#simply issuing the name of the function as the command will run the function
Enumerate-network
```

Arithmetic and typecasting

Performing mathematic operators like (+, and *) with numbers will result in a math function kind of like a calculator.

Doing this to strings, only concatenates

Ex:

```
$a= “amer”
```

$b= “ica”

$a+$b will equal “america”

$a+4 will equal “amer4”

$a*4 will equal “amerameramerамer”

4+4 will equal 8

“4”+”4” will equal 44 because you have made 4 a string by using quotes

Say you wanted to ended up with 2 strings of numbers against your will, but you wanted to add them as if they were integers…..You can typecast

$num1= “4”

$num2= “4”

[int]$num1+[int]$num2 will equal 8

This works in reverse as well.