Understanding the PowerShell Pipeline

TRAINSIGNAL

THE GLOBAL LEADER IN PROFESSIONAL COMPUTER TRAINING™

Pipeline Concepts

- Cmdlets work with objects
- Piped from one command to the next
- PowerShell displays remaining objects at the end of the pipeline
- Objects may change in the pipeline
- You get results only if there are objects in the pipeline

Write-Host vs. Write-Output

Write-Host skips the pipeline

- Writes to the console or hosting application
- Functionality might vary by host
  - `PS C:\> Write-Host $env:computername –foreground Green`
- Can't be redirected
- OK in scripts for progress messages
- NO OBJECTS

Write-Host vs. Write-Output

Write-Output writes the pipeline

- Preferred behavior
- Commands implicitly write pipelined output
- Test by piping expression to Get-Member

Watch for –Passthru

- Some commands don't write to the pipeline by default
- `PS C:\> Stop-Service wuauserv -passthru`

Pipeline Example

```
PS C:\> Get-Service

Status   Name               DisplayName
------   ----               -----------
Stopped  AeLookupSvc        Application Experience
Stopped  ALG                Application Layer Gateway Service
Stopped  AppIDSvc           Application Identity
Stopped  Appinfo            ication Information
Stopped  A                  n Management
Stopped  A                  ate Service
Running  A                  dio Endpoint Builder
Running  A                  dio
Stopped  AxInstSV           ActiveX Installer (AxInstSV)
Stopped  BDESVC             BitLocker Drive Encryption Service
Running  BFE                Base Filtering Engine
Running  BITS               Background Intelligent Transfer Ser...
```

Text based shells would need to parse this output

Pipeline Example

```
PS C:\> Get-Service | Where {$_.Status -eq "running"}

                            Displ
                            -----
                            Appli
Running  AudioEndpointBu... Windows Audio Endpoint Builder
Running  Audiosrv           Windows Audio
Running  BFE                Base Filtering Engine
Running  BITS               Background Intelligent Transfer Ser...
Running  cmdAgent           Bluetooth Support Service
Running  CmdVirth           COMODO Internet Security Helper Ser...
Running  ConfigFree Gadget  ConfigFree Gadget Service
Running  ConfigFree Service ConfigFree Service
Running  CryptSvc           Cryptographic Services
...
```

Get all Service objects

Only keep objects that are running

running running running running running stopped running stopped stopped stopped

Another Example

```
PS C:\>
```


Another Example

```
PS C:\> Get-Process
```

Get all processes

Another Example

```
PS C:\> Get-Process | Sort -property
Workingset –descending
```

Get all processes

Sort on the *workingset* property in descending order

Another Example

```
PS C:\> Get-Process | Sort -property
Workingset -descending | Select -First 10
```

Another Example

```
PS C:\> Get-Process | Sort -property
Workingset -descending | Select -First 10
| Export-Clixml c:\work\10Procs.xml
```

> PowerShell is all about the objects in the pipeline.

Using Variables

A variable is a place holder or container for a PowerShell object

- It "is" whatever it contains
- Can be a collection of objects

Create by assignment

- `PS C:\> $x = 123`
- `PS C:\> $scripts = dir c:\scripts\*.ps1`

Use variables as placeholders

- `PS C:\> $x*2`
  `246`
- `PS C:\> $scripts.count`
  `232`

Using Variables

- Variables are "point in time"
- Variable names contain letters, numbers, and _.
- Avoid spaces
- Give variables meaningful names
- The $ is used when referencing a variable

Subexpressions

Use parentheses to control pipelined execution

```
PS C:\> get-service wuauserv -computer (get-content
c:\work\computers.txt)
```

You can reference object properties as subexpressions

```
PS C:\> $svc = get-service browser

PS C:\> "The $($svc.displayname) service is $($svc.status)"
```

Other PowerShell Pipelines

Cmdlets must be designed to use these pipelines

Pipeline messages are controlled by preference variables
- DebugPreference (SilentlyContinue)
- ErrorActionPreference (Continue)
- VerbosePreference (SilentlyContinue)
- WarningPreference (Continue)

Redirecting Output

Using Out-File
- Use –Append to add to a file
- Specify encoding

Using Out-Printer
- Send to default printer
- Specify printer by name

Tee-Object
- Send to pipeline and a file
- Send to pipeline and a variable

Out* cmdlets should be at the end of your expression

```
PS C:\> dir –file -hidden | out-file –filepath
c:\work\rootfiles.txt
```


Stream Redirection

Redirect streams to Unicode text files

Stream	Value
Pipeline (Success)	1
Errors	2
Warning	3
Verbose	4
Debug	5

Write to file >
Append to file >>
Merge to file >&

Stream Redirection

```
PS C:\> get-wmiobject win32_logicaldisk
-comp "FOO","client2" 2>err.txt

PS C:\> c:\scripts\myscript.ps1 -verbose
2>err.txt 3>warn.txt 4>verbose.txt

PS C:\> c:\scripts\myscript.ps1 2>&1
1>data.txt
```

You can only merge to success stream

Sometimes the Pipeline Isn't the Right Plumbing

Cmdlets are designed to process groups of objects

Example: Get a bunch of something | Set a bunch of something

Sometimes you have to process objects individually

- Invoking a method on an object
- Incoming objects are of mixed types
- You want to do several things with each object

Enumerate with ForEach-Object

ForEach-Object

Alias is ForEach
…not to be confused with the ForEach enumerator

Do something for each piped in object

```
PS C:\> 2,4,8,16 | foreach { $_ * 3}
```

$_ indicates the current object in the pipeline

Can also use $psitem

ForEach Enumerator

- Similar to the VBScript construct
- For each item in a collection of items, do something with each item
- Define your own variable names
- Tend to use this more in scripting
- Does not write to the pipeline at the end

ForEach Enumerator

```
$files = dir c:\scripts –file
Foreach ($file in $files) {
   $fileage = ((get-Date) - $file.LastWriteTime)
  "$($file.name ) = $fileage"
}
```


ForEach Enumerator

```
$files = dir c:\scripts –file
Foreach ($file in $files) {
   $fileage = ((get-Date) - $file.LastWriteTime)
  "$($file.name ) = $fileage"
} | Out-File c:\work\fileage.txt
```

This will fail

ForEach Enumerator

```
$files = dir c:\scripts –file
Foreach ($file in $files) {
   $fileage = ((get-Date) - $file.LastWriteTime)
  "$($file.name ) = $fileage"
}
```


ForEach Enumerator

```
$files = dir c:\scripts –file
Foreach ($file in $files) {
   $fileage = ((get-Date) - $file.LastWriteTime)
  "$($file.name ) = $fileage" | Out-File …
}
```


Pipeline Demonstration

Lab

1. Get all running processes and save them to a variable called processes.

2. Write the contents of the processes variable to the pipeline

3. Pipe the processes variable to a text file

4. For each process in your variable take the workingset property and divide it by 1MB displaying the result.

5. Where do you think you could learn more about pipelines?