

Understanding the PowerShell Pipeline

TRAINSIGNAL
THE GLOBAL STANDARD 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

TRAINSIGNAL

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

TRAINSIGNAL

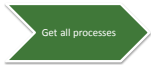
Another Example

```
PS C:\>
```

TRAINSIGNAL

Another Example

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



TRAINSIGNAL

Another Example

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



TRAINSIGNAL

Another Example

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



TRAINSIGNAL

Another Example

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



TRAINSIGNAL

“

PowerShell is all about the
objects in the pipeline.

”

TRAINSIGNAL

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

TRAINSIGNAL

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

TRAINSIGNAL

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)"
```

TRAINSIGNAL

Other PowerShell Pipelines

Error
Pipeline

Warning
Pipeline

Verbose
Pipeline

Debug
Pipeline

Cmdlets must be designed to use these pipelines

Pipeline messages are controlled by preference variables

- DebugPreference (SilentlyContinue)
- ErrorActionPreference (Continue)
- VerbosePreference (SilentlyContinue)
- WarningPreference (Continue)

TRAINSIGNAL

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
```

TRAINSIGNAL

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 >&

TRAINSIGNAL

Stream Redirection

```
PS C:\> get-wmiobject win32_logicaldisk  
-comp "F00", "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

TRAINSIGNAL

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

TRAINSIGNAL

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

TRAINSIGNAL

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

TRAINSIGNAL

ForEach Enumerator

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

TRAINSIGNAL

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

TRAINSIGNAL

ForEach Enumerator

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

TRAINSIGNAL

ForEach Enumerator

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

TRAINSIGNAL

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?

TRAINSIGNAL
