The Pipeline

Part 1

In this lecture we'll explore the pipeline, introduce some new commands and demonstrate how we can use the pipeline to complete everyday tasks. We'll also take a look at something **new called parameter binding.**

We'll talk about:

- What-if – With what-if, you can test your command before you try it.
- Send output to a file using out-file. **get-help *out-* Using out-file**
- Finally, we'll talk about how PowerShell sends commands through the pipeline.

Let's say we want to delete some files, but because we're using wildcards, we don't want to delete the wrong files. We can take the cautious approach and use the -whatif command.

Let's open explorer, go to our C drive

I have created **a folder** named **Company** with a subfolder named **HR.** I have created several txt files within each folder.

- **The goal** is to delete all the files in HR and Company without deleting any of the other files or folders on my c drive.

  First, I'll type the command then explain it

- Type **Get-ChildItem C:\Company\*.txt -Recurse | Remove-Item -WhatIf**

  Get-childitem – Is like using dir command

  I created two folders C:\Company and C:\Company\HR

  ***.txt – The files to look for**

  -Recurse – tells PowerShell to Search Subdirectories, here's our pipe operator.

  Remove-item – Equivalent to delete

  -whatif – Test but won't complete our command. Press Return.

Here we see whatif performing a test operation on these targets.

Now let's type the same command but this time we'll use the -confirm parameter

- **Get-ChildItem C:\Company\*.txt -Recurse | Remove-Item -confirm**

  PowerShell will ask you to confirm the deletion of each text file.

  Or we could go ahead and say Yes to all, and it will delete all the files. And we see that all our text files are gone.

Now let's check out the out-file command.

- **We're going to write the contents of our application log out to a file named app.txt**
- **Type get-eventlog -logname application | out-file c:\app.txt. press return**

  Now let's go to windows explorer and lets open the file in notepad. And as you can see there is the contents of the application log.

What command in PowerShell can we use to look at the **contents** of the application log file.

- **Type get-help *content* Press return, and we see that there is a get-content command**
- **Type get-content -path c:\app.txt press return. And our application log is displayed on the screen.**
- **Take a look at the headings, nice and neat.**

Let's go a little deeper into the pipeline. First let's review

- The **pipeline is used** for connecting commands together
- Using the pipeline, we can **pass the output of one command** to the **input of another command.**
- **The pipeline operator symbol can be found on most keyboards by holding down the shift key and pressing the key right above the enter key.**

So how does PowerShell know which commands will work together on the pipeline?

This time we'll use get-service.

- Go ahead and type **get-service -name bits | Stop-service press return**
- **BITS is the Background intelligent Transfer service. It is used to transfer files between computers.**
- **Get-service is command #1**
- **Stop-service is Command #2**

So, what actually happens when I run this command.

Get-service (1st command) is producing an object and then that object will go across the pipeline. Stop-service which is the 2nd command will connect to that object by mapping it to a **parameter.**

So how does PowerShell know what parameters to choose?

To make this happen there only four of pieces of information that PowerShell needs.

1. **Command #1 We'll look at the TypeName: (using get-member)**
2. **Can Command #2 use the same TypeName as Command #1 (use get-help)**
3. **Does Command #2 accept pipeline input (use get-help)**
4. **Can Command #2 accept pipeline input (ByValue or ByProperty)**

Open two PowerShell Windows

- For **command #1** type get-service -name **bits** | gm. **Press return, ok let's scroll up.**

  And we we see that our TypeName is ServiceController

- For **Command #2** type **get-help stop-service -full press return**

  Now let's scroll up until we find a **parameter** that **matches** the **TypeName**.

- So, for **command#2** we see that -InputObject matches the Typename from Command1 called ServiceController.

- **The next thing is, Does InputObject accept pipeline input and we see that's True.**

- Then, can InputObject **accept pipeline input byValue, and it does that as well**

Now let's run our command to verify that it will work

- **Type get-service bits | stop-service -passthru (will display an output) press return And we see that the service called Bits has stopped**

To get a clearer picture of parameter binding and how PowerShell accomplishes this let's review.

Let's take a look at **command 1** which is **get-service**

1. You need to use get-member to get the Typename. Which in this case is ServiceController. This is the name of the object that is being pushed across the pipeline.

2. Now for **command 2** which is **stop-service**, we open the help system and for the parameter -InputObject, we see three matches.
   The Typename, **which is service Controller**
   We see that it accepts pipeline input which is true
   and it accepts pipeline input = ByValue.
3. **ByValue** is the **first method** that PowerShell uses to move commands across the pipeline the second method is called **ByProperty**. We'll check that out in the next lecture.

BTW, if you ever plan on doing more than just running a few commands, understanding parameter binding for the PowerShell pipeline is essential, if you want to create your own tools.

Thanks for watching and we'll see you in the next lecture