Objects, Properties and Methods

Part 3 Properties

Now let's take a look at using the Properties of an object

- Let's use the **get-childitem** command. First type **get-childitem | gm PIPE OPERATOR (hold your shift key down and press the key right above the enter key**
  Then Type GM press enter

  We want to use **the creationtime property** to find out the **creation date** of the **current** version of PowerShell, press return

  Let's type the command then I'll explain it.

  Type (Get-ChildItem $pshome\PowerShell.exe).creationtime

- **The most common way** to get the values of the properties of an object is to

  use the **dot method**. That means that you **first surround** the parameter and the path with parenthesis. Then insert a (.) then the property. Which in this case is **creationtime.**
  Now press return

  And **we get Wednesday**, April 11, 2018 07:35 PM

- By the **way $pshome** is the path to the PowerShell home folder

Another way to **get the properties of an object** is to use the **select-object command**. The **select-object** command has a **parameter called –property** that will **get the properties of an object.**

```
Syntax
    Select-Object [[-Property] <Object[]>] [-ExcludeProperty <String[]>] [-ExpandProperty <String>] [-First <Int32>] [-
InputObject <PSObject>] [-Last <Int32>] [-Skip <Int32>] [-Unique ] [-Wait ] [<CommonParameters>]

    Select-Object [[-Property] <Object[]>] [-ExcludeProperty <String[]>] [-ExpandProperty <String>] [-InputObject
<PSObject>] [-SkipLast <Int32>] [-Unique ] [<CommonParameters>]

    Select-Object [-Index <Int32[]>] [-InputObject <PSObject>] [-Unique ] [-Wait ] [<CommonParameters>]
```

Let's type help Select-object -showwindow and let's analyze the syntax.

- **The parameter** that we are going to **be using is called -property.**

  Select-object is the name of the **cmdlet.**

  Notice that the **parameter -property** and the **argument's value type - called object,**

both are **surrounded by square brackets**. That means that **both parameter and argument** are **optional and not needed**.

Notice also that **-property** is surrounded by a **separate set of square brackets so -property is positional as well**.

We can verify that by **scrolling up** and looking at the **parameter attributes**. Which **tell us** that **-property has a position of 0,** and it is **not required**.

That tells us that **-property should be located in the first position** in the lineup of parameters.

We can also see that the **argument** has **two square brackets** inside the **two angle** brackets this means that the **parameter -property can take multiple arguments separated by a comma**.

- **We'll use get-eventlog security and select-object for this demonstration**

  Instead of displaying the whole security log let's just display the **newest 6 events.**

  Type Get-eventlog -logname security -newest 6

- Now let's use **Get-member, which** will show us the **properties and methods**

  Type Get-eventlog -logname security -newest 6 | get-member

- Let's **select** a few useful properties. How about **Time-Generated, EventID and machinename**

  Type Get-eventlog -logname security -newest 6 | Select-object Source, TimeWritten, machinename, Message. And press return.

  And we see that the command ran.

  In case your'e wondering why I used -logname, because as we discussed earlier it's optional and not needed. Here's why, when your'e first starting out in PowerShell, you might want to go ahead and type out some of the optional parameters. Especially if you plan on saving your commands and one liner's as scripts for later use. It just makes it easier to remember what these commands and parameters are doing if you go ahead and type them out. The same goes for aliases you can use gsv for get-service . But it's a whole lot easier to remember what get-service -name BITS is doing instead of gsv bits. Both commands will work, but when you've type out the whole command it's easier to understand especially when you're first starting with PowerShell.

  Thanks for watching and we will see you in the next lecture.