# PowerShell Lab 1

Introduction to PowerShell

## Update Help

PowerShell is installed with the operating system on Windows 7 and up. Usually, the complete help is not installed, so we’ll need to remedy that. Run PowerShell from an elevated command prompt (right-click, run as administrator) and type this command:

Update-Help

## Tabbing through commands

Type help get-ev and hit tab until you get help Get-EventLog

What does Get-EventLog do?

## More help

Use help \*common\* to see help objects that contain the word common. Then enter the help command from that list to see what common parameters are.

Scan through the results of your command to see what the WhatIf parameter does.

## ISE and show-command

Type ISE at the PowerShell prompt. This will open the Integrated Scripting Environment. The white part on top is where you would type a script. The blue part at the bottom is a command window where your results appear. You can also type commands there. Start typing get-process and see what happens. Also note the window at the right that shows all the commands. At the right side of the screen, you should see the Commands Add-on. If not, select View and put a check in Command Add-on. In the command add-on, select a command you have seen in the Cyber Aces modules, Get-Alias for example. The add-on will show you the parameters and choices you have available and allow you to enter data into them. When you click ok, the results are put back in the command line.

From a regular PowerShell prompt (not ISE) type Show-Command. This gives the same Command Add-on you just saw in the ISE.

## PowerShell Providers

A file system has a hierarchical structure that you are accustomed to by now. PowerShell treats other parts of the operating system that have hierarchical structures as though they are file systems as well. PowerShell calls these “providers”. Providers look like drives on the file system, and many of them can be manipulated by commands as if they were files. This is why some commands have curious names. The formal name for dir is Get-ChildItem, since it does more than get a directory of the file system. Likewise, cd is really Set-Location since it does more than change directories.

## Registry

The registry is a PowerShell provider. Besides being able to use cd and ls (another alias for Get-ChildItem) in the file system, you can use them to navigate the registry. Type:

cd HKLM: (the long version would be Set-Location HKLM: )

Use ls and cd to look around--you will get a few access errors when you look at keys you don’t have permission to see, but that's ok. When you are done, type cd c: to get back to the file system.

## Environment variables

Type ls env: and see what you get. Since Powershell treats environment variables as a provider, you can view or change environment variables easily. To see just the Path variable, you could enter ls env:Path. To create or change environment variables, treat Env: like a variable and put a $ in front of it to get $Env:. To make a new environment variable called Boo with a value of Just Kidding you would enter  
$env:Boo = "Just Kidding"

See <https://www.tachytelic.net/2019/03/powershell-environment-variables/>

## Other providers

Enter Get-PSDrive and see what you get. This shows all the other providers that are available on your system. These are items that have hierarchical structure and can be accessed like the file system. In a computer with more tools installed you can see Microsoft Active Directory, SQLserver, Exchange, and many other systems.

## Get-Process and a little about objects

Run the Get-Process command and note that it shows all processes, plus select info about them. (More info is available, there's just not room for it all on the screen.)

Open 2 or 3 notepads and 2 or 3 calculators

Enter Get-Process. You’ll find lots of stuff there, but you should be able to find your notepad and calculator processes. It looks like multiple instances of notepad create separate processes, but instances of calculator run under one process. Interesting.

The Get-Member command is very useful for finding the methods (actions you can take) and properties (information you can use) that are passed by the object(s) that a command creates. For example, let’s see what is available in the object(s) created by Get-Process.

Get-Process | Get-Member

What are some of the actions (methods) you can take against an object created by Get-Process?

The properties of Get-Process objects can tell you just about anything you could want to know about a process, including whether it is running, when it started, windows it has open, memory it uses, and more.

Type get-process -n and hit tab until you have get-process -name

Finish the command, to make it get-process -name notepad

If you want to know where the notepad executable is, just type

(get-process -name notepad).path or

Get-Process -Name notepad | Format-List -Property path

(Format-List (usually known by its alias fl) is a command that outputs to the screen.)

Now try it for calculator...

Now try get-process -name calculator | stop-process. What do you think will happen? Do the same thing for notepad.

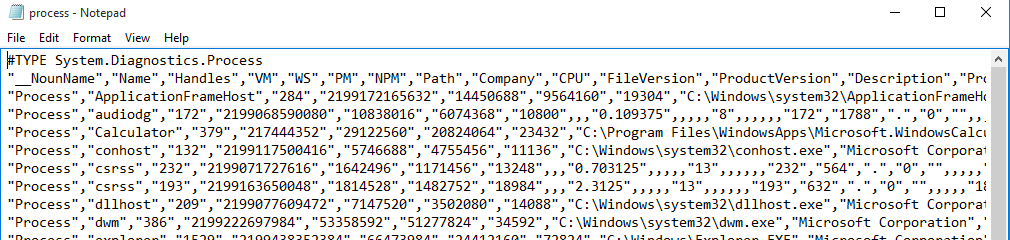
What would happen if you ran this from a prompt with admin rights? (Note: don’t do this unless you are feeling adventurous. It would be best if you ran this on a virtual machine, and not your host.)

get-process | stop-process

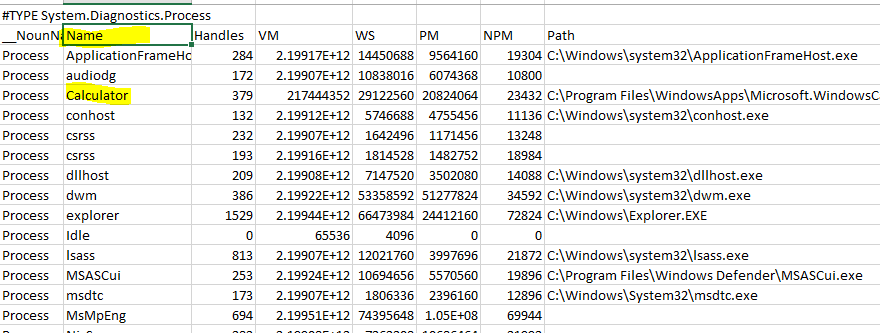
## More on Objects and Properties

When you run Get-Process the result will be one object for each process. Each of those objects will have several properties describing the object (in this case, a process.) To make it easier to visualize, change directory to your desktop and then run the following command. Export-Csv will format the output in a text file of Comma Separated Values (CSV).

Get-Process | Export-Csv process.csv

If you open the resulting file, you’ll see that the first line describes the type of object you saved, a Process object in this case. The next line is a list of the property names. After that, there is one line for each process (object) and the values for the properties of that object.

If you open the file in a spreadsheet, you’ll see this.



Each row is an object. For example, there is a row for the Calculator process. Each column specifies one property for each of the processes. In the spreadsheet, you can see the Path for each process object. The names for some of the other properties are cryptic. You can get an idea of what some of them are by piping Get-Process into Get-Member. For more information, you’ll need to resort to a search engine.

Now that we’ve looked at objects and processes the old-fashioned way (rows and columns in a spreadsheet,) let’s use the built-in PowerShell way. Execute the command,  
Get-Process | Out-GridView  
Compare it to the spreadsheet output.

## Exercise

Pipe Get-ChildItem (ls or dir are aliases) into Get-Member (gm) and examine the methods and properties. Pipe Get-ChildItem into Export-Csv and open the resulting file in a spreadsheet, or pipe Get-ChildItem into Out-GridView. Examine the objects and their properties.

1 ) Hand in a screenshot of your spreadsheet/GridView.

## Seeing the properties you want to see

Compare the list of the properties you saw from Get-Member or the spreadsheet you made from Get-ChildItem with what you see if you just run Get-ChildItem (or ls) at the prompt. Many of the properties are not shown by default in display output. An easy way to see the other properties is to pipe the output into Format-Table (usually ft) or Format-List (fl) and list the properties you want or use \* to see all of them. Listing the properties you want works with many commands, including Export-Csv.

Look at the list of properties of Get-ChildItem or Get-Process. Pick some that aren’t displayed with the basic command, and show them with ft and fl. For example,

ls | ft -Property Name, CreationTime, LastAccessTime, LastWritetime

2) Hand in a screenshot of your output.

## Import-Csv

A nice thing about CSV files is that they save our data in a form that maintains the object format. You should already have files that save the output from Get-Process and Get-ChildItem. Use one of the commands below to move the data back into PowerShell’s memory.

$processObj = Import-Csv <filename you used to save Get-Process>  
or  
$dirObj = Import-Csv <filename you used to save Get-ChildItem>

Then look at the data with commands such as

$processObj.Path  
$processObj | ft -Property path  
or  
$dirObj.Name  
$dirObj | ft -Name

3) Hand in a screenshot of one of these commands.

Hand In  
Hand in the screenshots from items 1, 2, and 3 above.