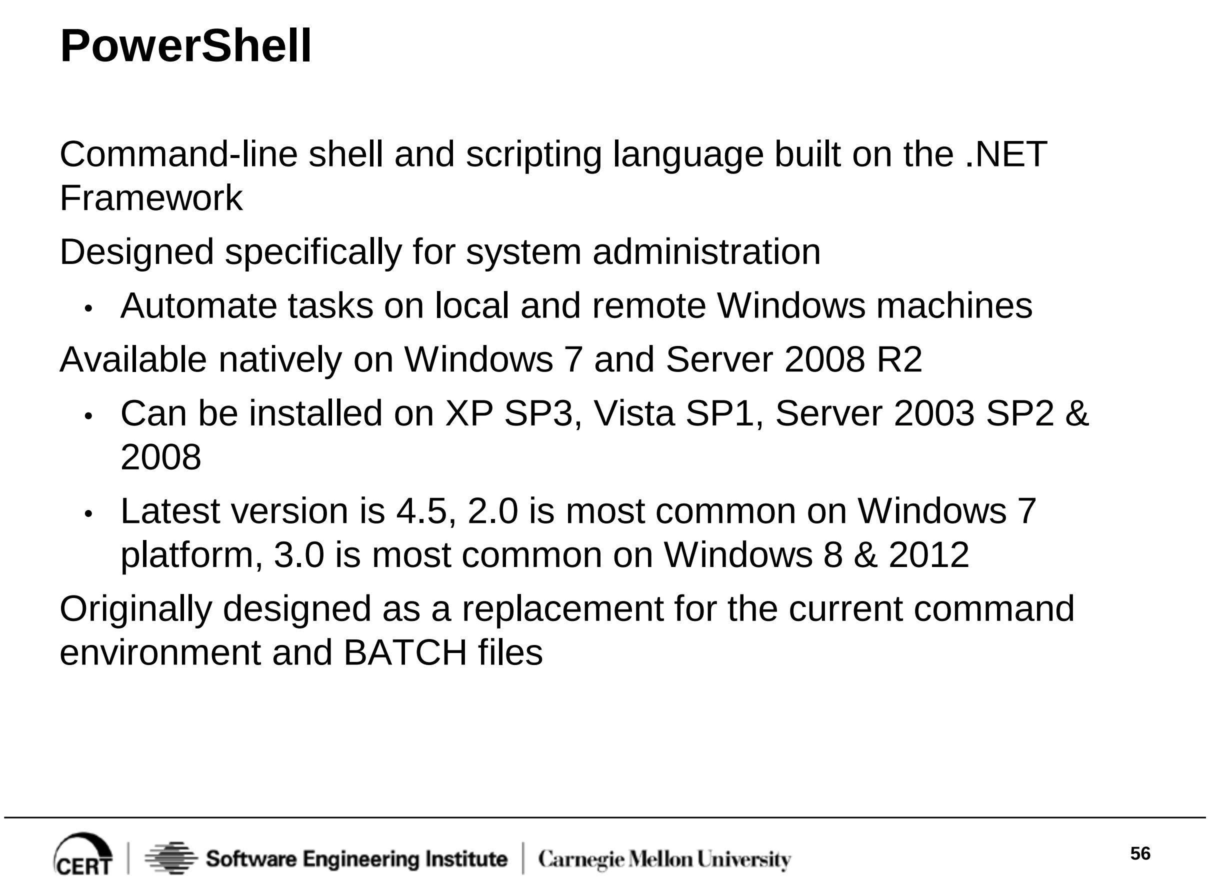
PowerShell Commands

Table of Contents

PowerShell



PowerShell

Command-line shell and scripting language built on the .NET Framework

Designed specifically for system administration

- Automate tasks on local and remote Windows machines

Available natively on Windows 7 and Server 2008 R2

- Can be installed on XP SP3, Vista SP1, Server 2003 SP2 & 2008
- Latest version is 4.5, 2.0 is most common on Windows 7 platform, 3.0 is most common on Windows 8 & 2012

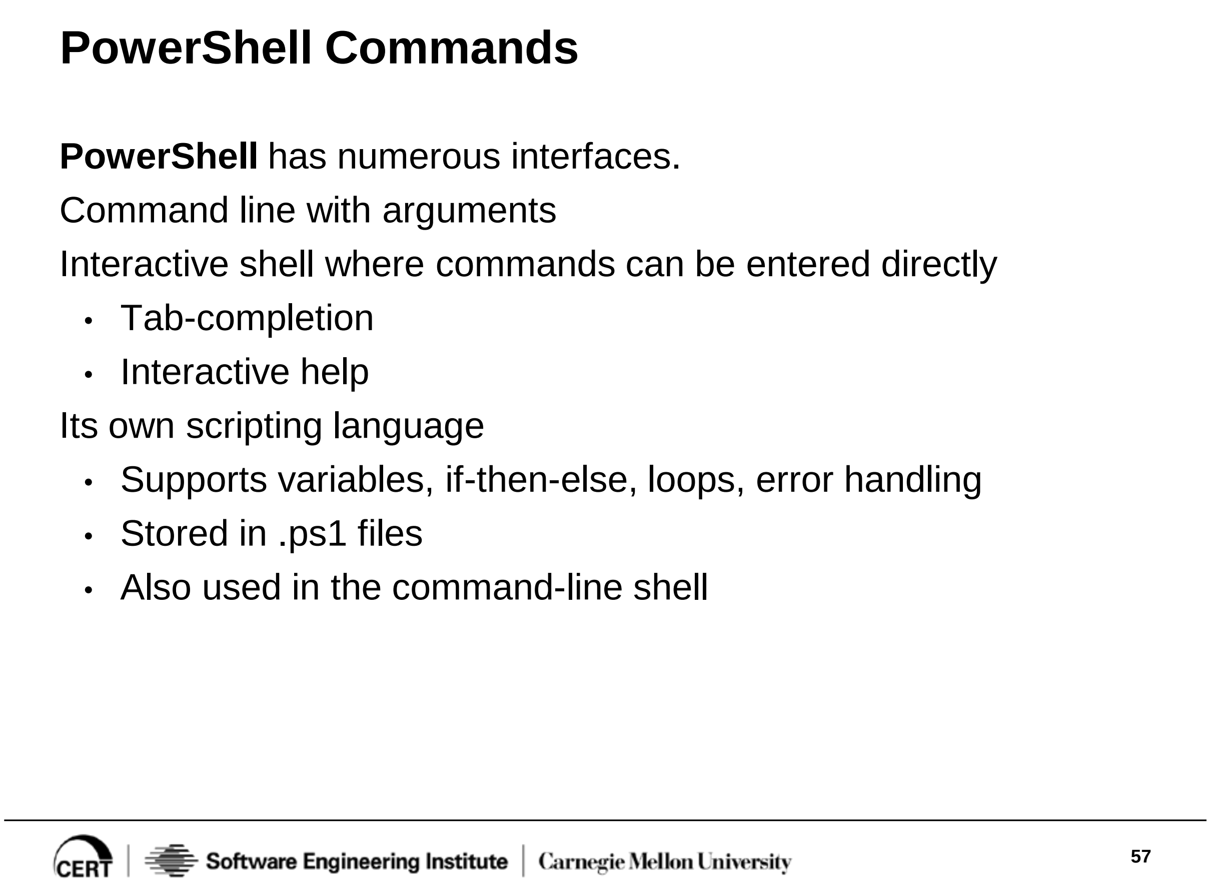
Originally designed as a replacement for the current command environment and BATCH files

CERT | Software Engineering Institute | Carnegie Mellon University 56

**056 Instructor: So we'll go ahead and continue with PowerShell. This is a relatively new-- I shouldn't say new, but compared to the regular command line shell in the scripting language-- that's built on the .NET framework. It's a very powerful tool, designed specifically for system administration. So again, you're seeing now a lot of similarities with the net and being able to use information from WMIC and now with PowerShell.

It automatically the tasks, so it makes them easier for administrators to take care of multiple machines across the network, and it's available on Windows 7 and Windows 8 and newer, and it can be installed on XP, Windows XP, and the Vista as well, and Windows 8, and I presume Windows 10 will probably have this on there. It's a very powerful tool. And I guess it was originally designed to replace the current command environment and batch scripting support, but as you know, we still have batch around today. So must still be okay, and a lot of people probably still use it.

PowerShell Commands



PowerShell Commands

PowerShell has numerous interfaces.

Command line with arguments

Interactive shell where commands can be entered directly

- Tab-completion
- Interactive help

Its own scripting language

- Supports variables, if-then-else, loops, error handling
- Stored in .ps1 files
- Also used in the command-line shell

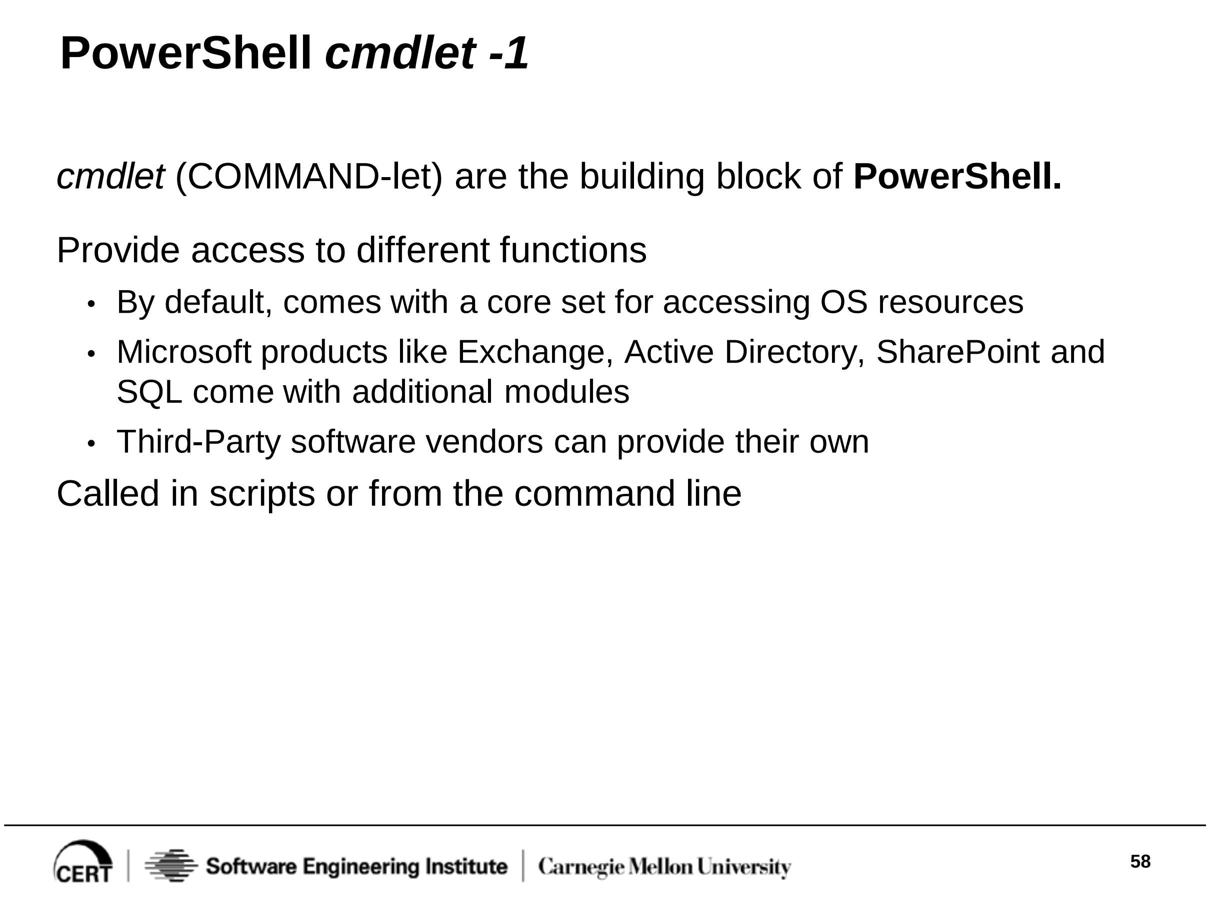
CERT | Software Engineering Institute | Carnegie Mellon University

57

**057 PowerShell has quite a few interfaces. The interactive shell, where the commands are entered--

it's kind of nice you have the tab completion. If you've ever used Linux-- well, actually, even in Windows-- on the command line, if you start spelling a word you can hit Tab and it'll finish the line for you, and it does have an interactive Help. It has its own scripting language that does support variables-- if-then-else, some looping, and there is error handling. And then it's stored in ps1 files, or .ps1 files, and then it can be used in command line shell.

PowerShell cmdlet -1



PowerShell *cmdlet -1*

cmdlet (COMMAND-let) are the building block of **PowerShell.**

Provide access to different functions

- By default, comes with a core set for accessing OS resources
- Microsoft products like Exchange, Active Directory, SharePoint and SQL come with additional modules
- Third-Party software vendors can provide their own

Called in scripts or from the command line

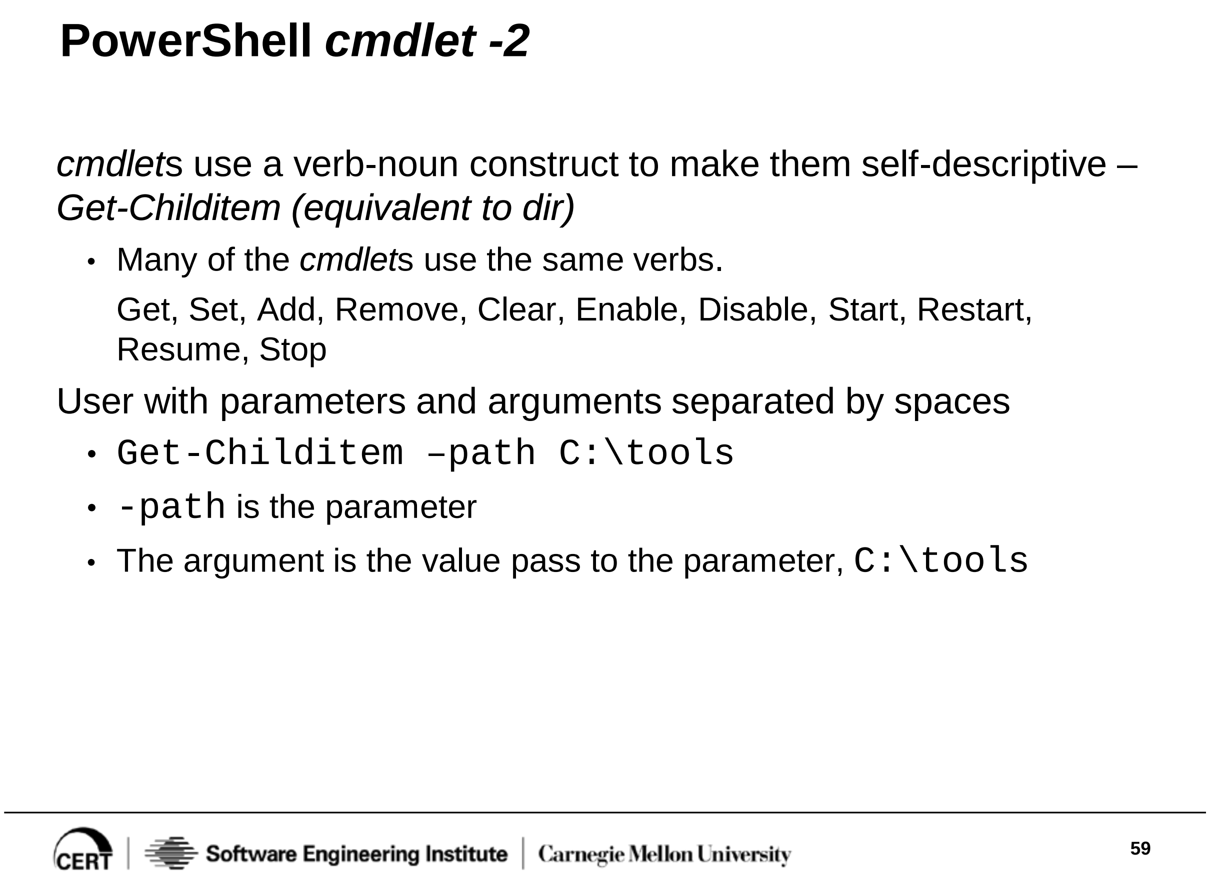
58

**058 PowerShell has something called "command-lets," even though it's kind of spelled C-M-D-let, and this is the main building block used in PowerShell. It gives you access to all the functionality, or a lot of functionality. There's a core set of them for accessing the operating system resources, and Microsoft has the products like Exchange, Active Directory, SharePoint and SQL come with some additional modules that you can access through PowerShell. So it's even better for being able to administer and to do other things on other devices that have like Exchange and Active Directory on them.

Also nice is that there's third-party software vendors that can also provide their own cmdlets that can reach out to their own machines and their own stuff, so they can kind of customize that for themselves.

And then you can call it in scripts and also from the command line, because it pops up its own little shell when you type in PowerShell.

PowerShell cmdlet -2



PowerShell *cmdlet -2*

*cmdlet*s use a verb-noun construct to make them self-descriptive – *Get-Childitem (equivalent to dir)*

- Many of the *cmdlet*s use the same verbs.
  Get, Set, Add, Remove, Clear, Enable, Disable, Start, Restart, Resume, Stop

User with parameters and arguments separated by spaces

- `Get-Childitem –path C:\tools`
- `-path` is the parameter
- The argument is the value pass to the parameter, `C:\tools`

CERT | Software Engineering Institute | Carnegie Mellon University 59

**059 So PowerShell cmdlets use a verb-noun construct. So like the Get-Childitem-- it's kind of nice for readability and understanding what it does. So some of the verbs are the get, set, add, remove-- pretty intuitive, pretty straightforward-- enable and disable-- those kind of things.

The parameters and arguments. You separate by space. So you do a Get-Childitem and then the -path. Path is the parameter and then the argument is the value pass to the parameter c:\tools.

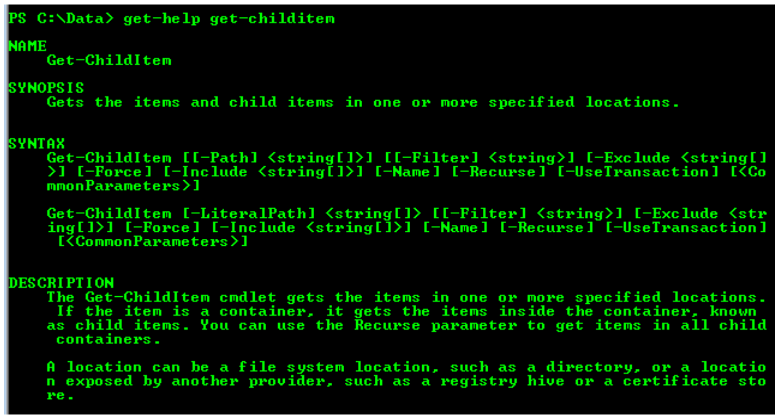
PowerShell Using cmdlet -1

PowerShell Using *cmdlet -1*

Start **PowerShell** by typing `powershell`

- Looks like command prompt, but with PS in front
- For help use: `get-help <`*`cmdlet name`*`>`
- For help on all the possible get commands use:

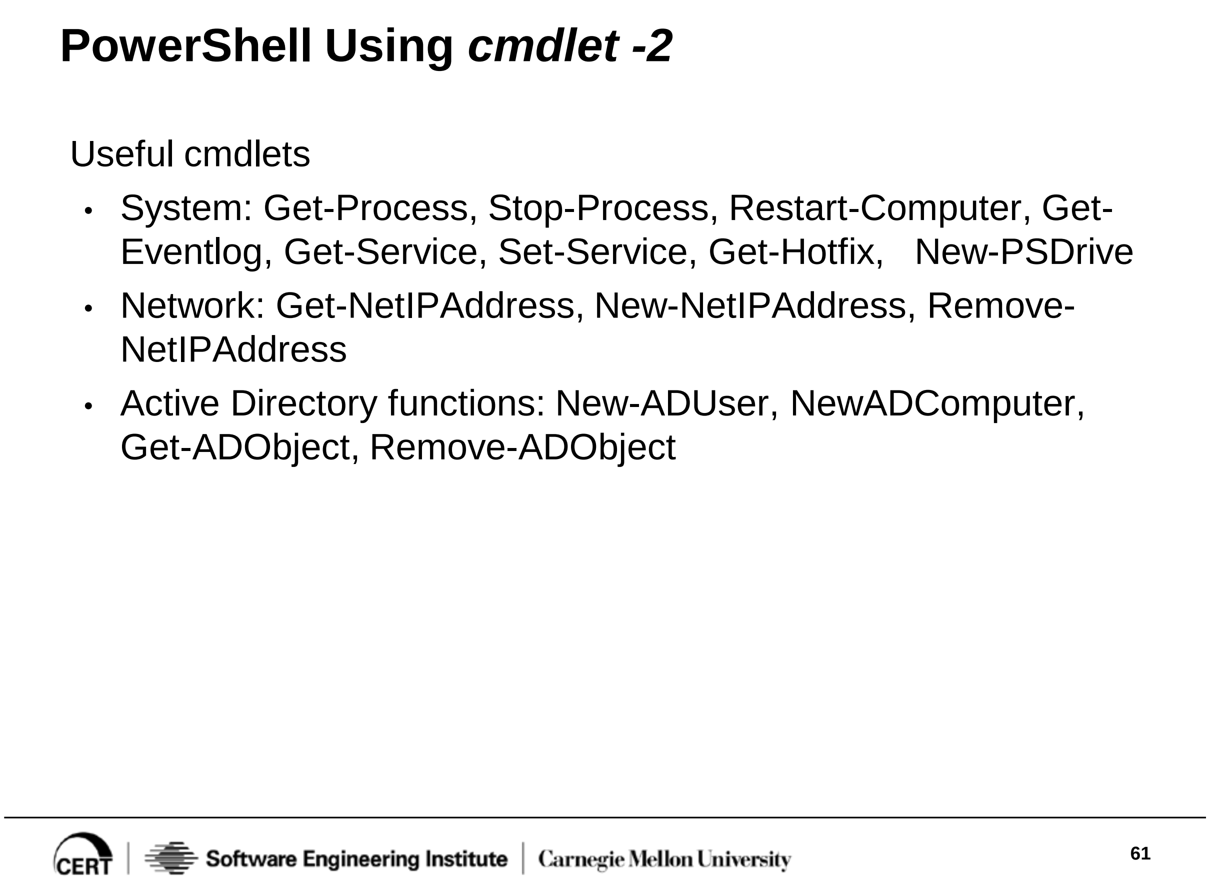
  `get-help –Name get-*`



```
PS C:\Data> get-help get-childitem

NAME
    Get-ChildItem

SYNOPSIS
    Gets the items and child items in one or more specified locations.


SYNTAX
    Get-ChildItem [[-Path] <string[]>] [[-Filter] <string>] [-Exclude <string[]
    >] [-Force] [-Include <string[]>] [-Name] [-Recurse] [-UseTransaction] [<Co
    mmonParameters>]

    Get-ChildItem [-LiteralPath] <string[]> [[-Filter] <string>] [-Exclude <str
    ing[]>] [-Force] [-Include <string[]>] [-Name] [-Recurse] [-UseTransaction]
     [<CommonParameters>]


DESCRIPTION
    The Get-ChildItem cmdlet gets the items in one or more specified locations.
     If the item is a container, it gets the items inside the container, known
    as child items. You can use the Recurse parameter to get items in all child
     containers.

    A location can be a file system location, such as a directory, or a locatio
    n exposed by another provider, such as a registry hive or a certificate sto
    re.
```

**060 And like I was saying, you just start it up by typing PowerShell in, and then it opens up with a little PS in front there so you kind of know you're in a different shell than the regular command prompt that you start with. You can do a get-help, so that's that verb-noun, and then you can do the cmdlets, and if you want a list of all possible commands you can do the get-help -Name get-star, another-- you can get at all of those using that context.

PowerShell Using cmdlet -2



PowerShell Using *cmdlet -2*

Useful cmdlets

- System: Get-Process, Stop-Process, Restart-Computer, Get-Eventlog, Get-Service, Set-Service, Get-Hotfix,   New-PSDrive
- Network: Get-NetIPAddress, New-NetIPAddress, Remove-NetIPAddress
- Active Directory functions: New-ADUser, NewADComputer, Get-ADObject, Remove-ADObject

61

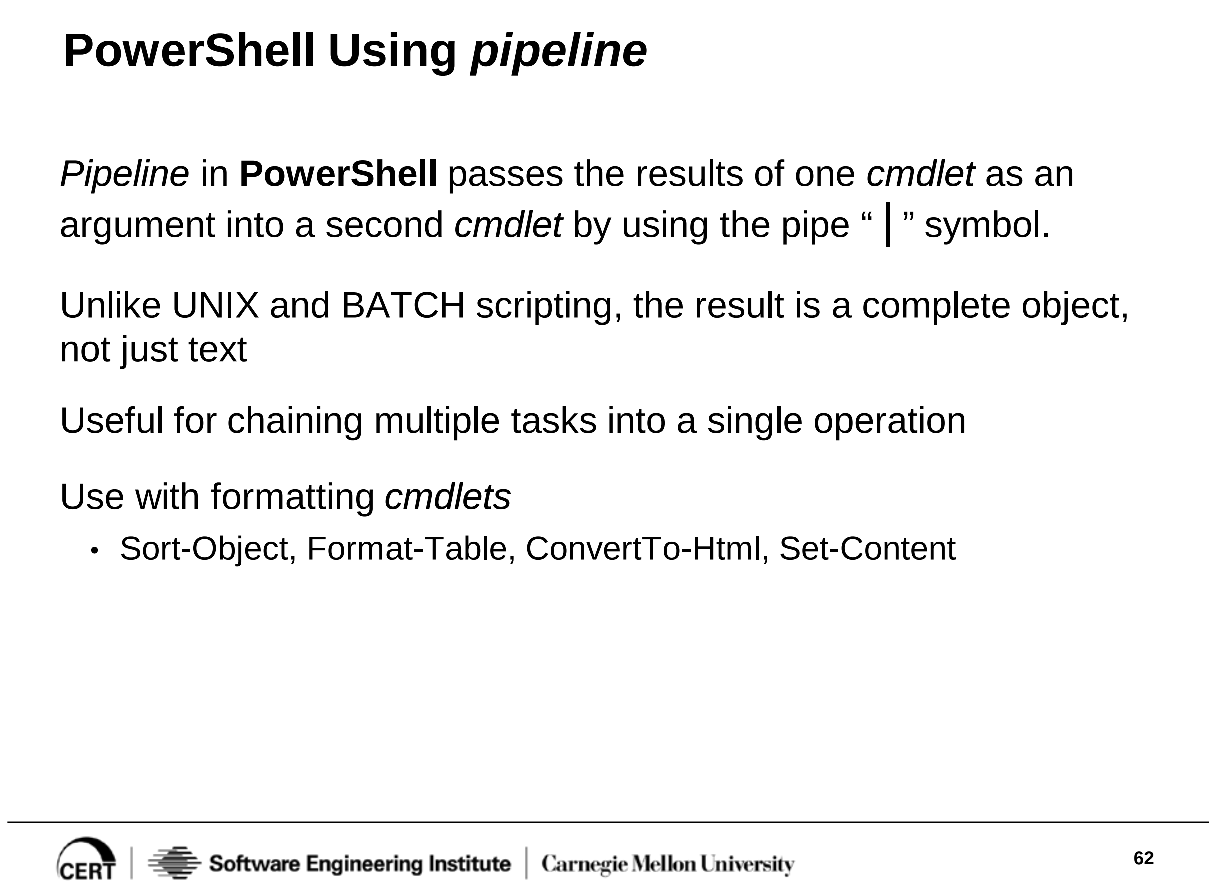
**061 And here are some of the more useful cmdlets-- Get-Process, Stop-Process-- right? Some pretty straightforward-- it's real nice to be able to read this. You can kind of tell exactly what it is you're able to get or request. So you can do Get-NetIPAddress. So instead of doing an ipconfig like you would in a regular command shell or command prompt, you would do the Get-NetIPAddress. And again, it is a little bit of typing, but once you start on it, if you hit the tab, they tend to be able to finish up for you. So that's kind of nice there. They're a little wordy in some ways, in some of them.

And then Active Directory functions,
you have access to that, and that's
not always easy to directly access.
So that's very helpful. Any questions?

Student: Can you call some
of the other functions like string?
Like net use where-- can you create
subroutines and call them with
PowerShell?

Instructor: My understanding is
you can access the regular command
line commands as well as the
PowerShell commands. Yes. So you
can do that.

PowerShell Using pipeline



PowerShell Using *pipeline*

Pipeline in **PowerShell** passes the results of one *cmdlet* as an argument into a second *cmdlet* by using the pipe “ | ” symbol.

Unlike UNIX and BATCH scripting, the result is a complete object, not just text

Useful for chaining multiple tasks into a single operation

Use with formatting *cmdlets*

- Sort-Object, Format-Table, ConvertTo-Html, Set-Content

CERT | Software Engineering Institute | Carnegie Mellon University

62

**062 So PowerShell allows passing of information, just like the other-- or the standard commands that you can get at through the Windows command line using the pipe-- they kind of call that a pipeline here, if you think of it as a pipeline, entering out one side and coming out the other. It's the same vertical or pipe symbol that you use.

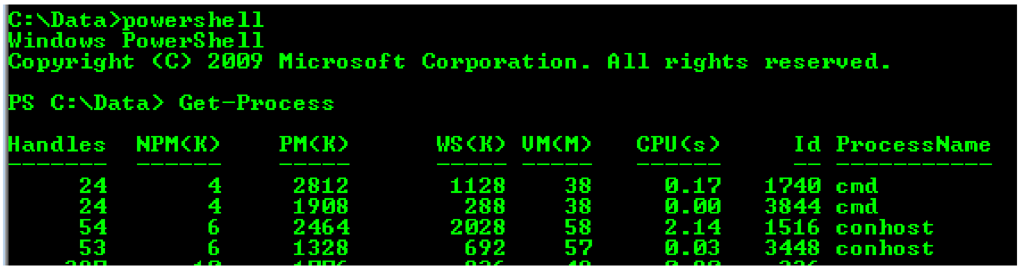
Unlike UNIX and batch scripting, however, the result is a complete object, so it's not just text that shows up. You actually get a-- you're sending an object. And it's very useful because you can chain multiple tasks together in one operation and you can just keep using that object, so you don't have to keep calling it.

And there are some formatting cmdlets here-- Sort-Object, Format-Table and the like.

PowerShell Examples -1

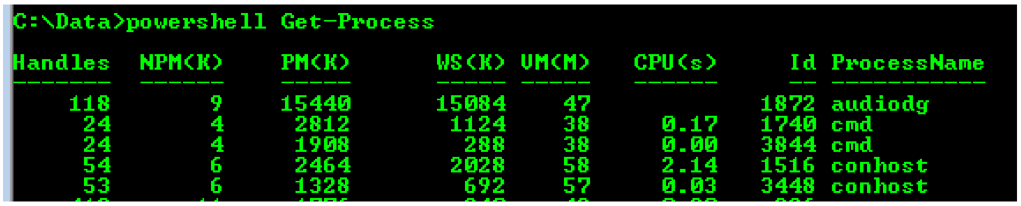
PowerShell Examples -1

Most of these examples can be run from inside PowerShell or from the command-line by prefixing it with PowerShell.



```
C:\Data>powershell
Windows PowerShell
Copyright (C) 2009 Microsoft Corporation. All rights reserved.

PS C:\Data> Get-Process

Handles  NPM(K)    PM(K)      WS(K) VM(M)   CPU(s)     Id ProcessName
-------  ------    -----      ----- -----   ------     -- -----------
     24       4     2812       1128    38     0.17   1740 cmd
     24       4     1908        288    38     0.00   3844 cmd
     54       6     2464       2028    58     2.14   1516 conhost
     53       6     1328        692    57     0.03   3448 conhost
```

VS



```
C:\Data>powershell Get-Process

Handles  NPM(K)    PM(K)      WS(K) VM(M)   CPU(s)     Id ProcessName
-------  ------    -----      ----- -----   ------     -- -----------
    118       9    15440      15084    47            1872 audiodg
     24       4     2812       1124    38     0.17   1740 cmd
     24       4     1908        288    38     0.00   3844 cmd
     54       6     2464       2028    58     2.14   1516 conhost
     53       6     1328        692    57     0.03   3448 conhost
```

CERT | Software Engineering Institute | Carnegie Mellon University 63

**063 And here's some example PowerShell displays. Here's the Get-Process, and this looks like perhaps--

Student: They're different.

Instructor: Actually, this is run from inside the PowerShell command prompt with the PS in front, and this is just run with using PowerShell as a prefix to the actual word. So essentially they're saying they're different in the sense that whatever processes popped up happen to be different on here, but they're just showing you there's two different ways you can call it. One is inside the PowerShell shell, and then the other is just using the word PowerShell as a prefix to what it is you're trying to reach out and get.

Instructor 2: I was just going to say here that the first screen you see here, you're actually running inside the PowerShell command prompt, for lack of a better word. The second one is where you would actually use this in a batch file. So if you wanted to script a PowerShell command, you could do that. Actually, you could do it within the PowerShell command line too, but if you wanted to add a PowerShell cmdlet to a Windows batch file or something like that, you would use the second version of this. It's calling the exact same command. You would get the same output if you ran it at the same time. But the bottom one is useful for scripting in batch files. The top one is really just for interactive.

Instructor: Right. Did everybody get that?

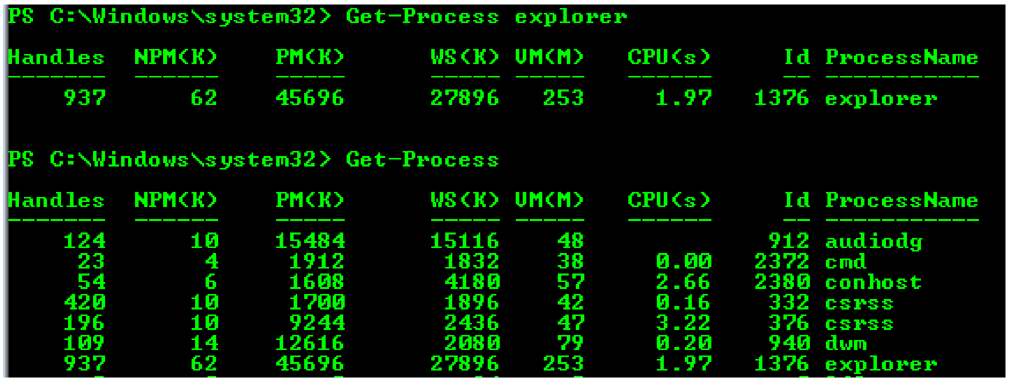
PowerShell Examples -2

`Get-Process explorer`

- Lists all the processes running named explorer, by itself will list all processes

`Stop-Process –processname explorer`

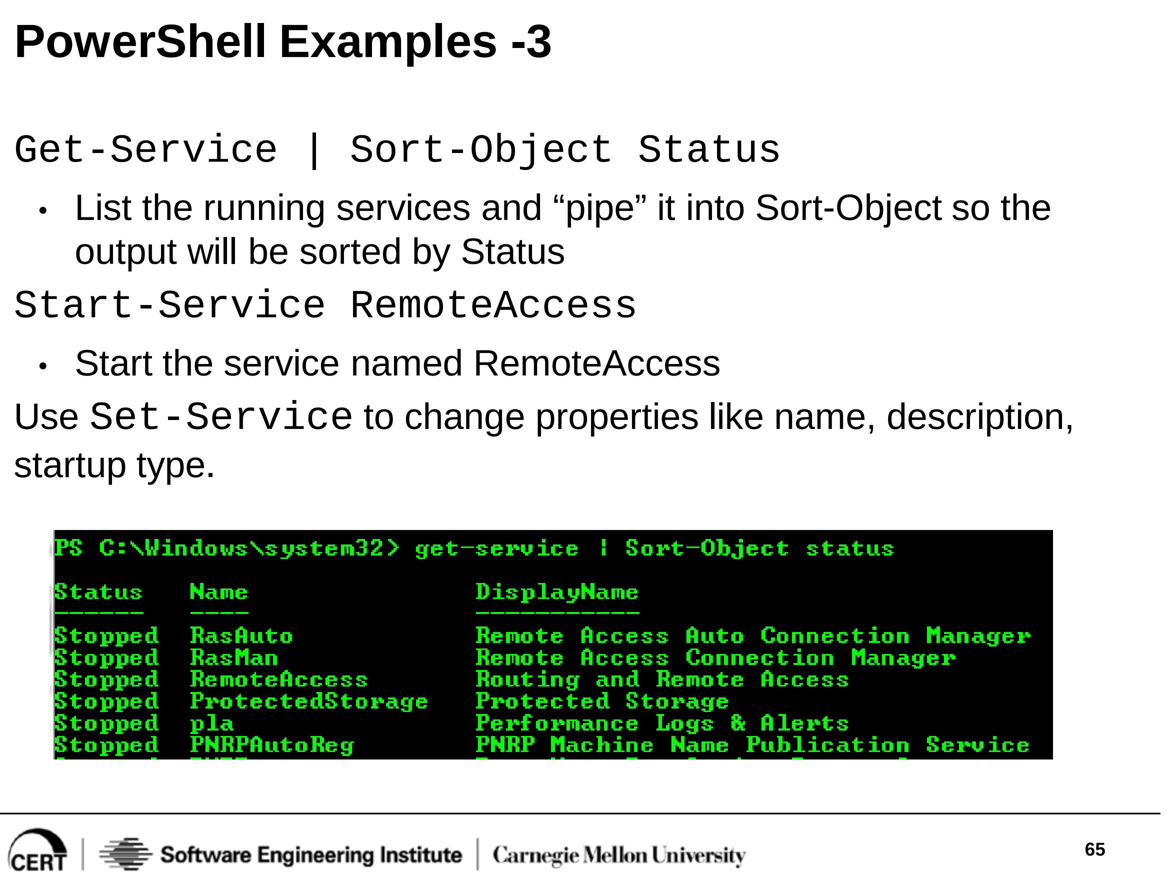
- Stop the process named explorer, can also specify the process ID



```
PS C:\Windows\system32> Get-Process explorer

Handles  NPM(K)    PM(K)      WS(K) VM(M)   CPU(s)     Id ProcessName
-------  ------    -----      ----- -----   ------     -- -----------
    937      62    45696      27896   253     1.97   1376 explorer


PS C:\Windows\system32> Get-Process

Handles  NPM(K)    PM(K)      WS(K) VM(M)   CPU(s)     Id ProcessName
-------  ------    -----      ----- -----   ------     -- -----------
    124      10    15484      15116    48             912 audiodg
     23       4     1912       1832    38     0.00   2372 cmd
     54       6     1608       4180    57     2.66   2380 conhost
    420      10     1700       1896    42     0.16    332 csrss
    196      10     9244       2436    47     3.22    376 csrss
    109      14    12616       2080    79     0.20    940 dwm
    937      62    45696      27896   253     1.97   1376 explorer
```

**064 So if you just type in the Get-Process explorer, it'll list all your processes-- named explorer, so you can kind of give it a title and it'll go search for a specific item. And then of course the power being that you can reach out and stop processes, not unlike WMI or the net commands. As long as you have enough privileges, you can do the same thing here where you can stop process and the process name is explorer, and you can also specify the ID. So if you know the ID is 1376-- so instead of putting explorer up there-- and I'm not sure if it's process name though at that point. It might be process dash process ID and then put this 1376-- you'll be able to shut down that process.

PowerShell Examples -3



PowerShell Examples -3

`Get-Service | Sort-Object Status`

- List the running services and “pipe” it into Sort-Object so the output will be sorted by Status

`Start-Service RemoteAccess`

- Start the service named RemoteAccess

Use `Set-Service` to change properties like name, description, startup type.

```
PS C:\Windows\system32> get-service | Sort-Object status

Status   Name               DisplayName
------   ----               -----------
Stopped  RasAuto            Remote Access Auto Connection Manager
Stopped  RasMan             Remote Access Connection Manager
Stopped  RemoteAccess       Routing and Remote Access
Stopped  ProtectedStorage   Protected Storage
Stopped  pla                Performance Logs & Alerts
Stopped  PNRPAutoReg        PNRP Machine Name Publication Service
```

CERT | Software Engineering Institute | Carnegie Mellon University

65

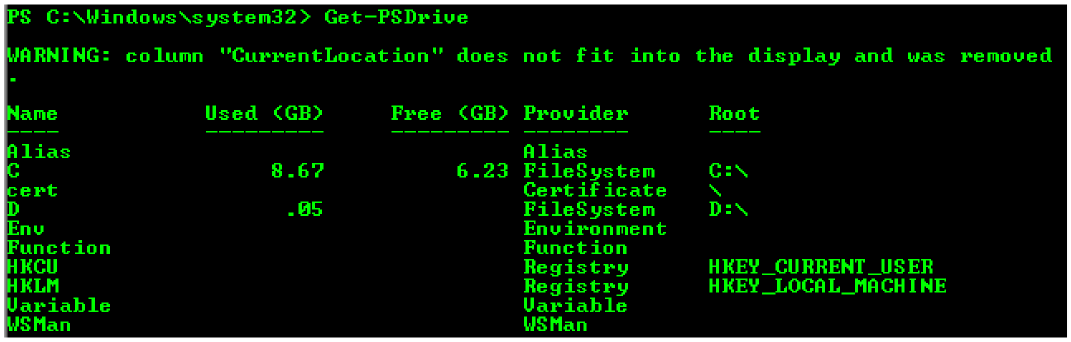
**065 So, moving along. Get-Service, and you can pipe that over to Sort-Object based on status. It'll list the running services-- and-and-- sorry about that. I have to remember-- and then pipe it over, and the output will be sorted by the status, as you can see here.

Then Start-Service RemoteAccess. That'll bring up the remote access. And then Set-Service to change the properties, like the name and the description and the startup type.

PowerShell PSDrive -1

Use `PSDrive` to navigate the registry like a file system.

- `Get-PSDrive` will show a list of current drives and aliases in this PowerShell session.



```
PS C:\Windows\system32> Get-PSDrive

WARNING: column "CurrentLocation" does not fit into the display and was removed
.

Name           Used (GB)     Free (GB) Provider      Root
----           ---------     --------- --------      ----
Alias                                  Alias
C                   8.67          6.23 FileSystem    C:\
cert                                   Certificate   \
D                    .05               FileSystem    D:\
Env                                    Environment
Function                               Function
HKCU                                   Registry      HKEY_CURRENT_USER
HKLM                                   Registry      HKEY_LOCAL_MACHINE
Variable                               Variable
WSMan                                  WSMan
```

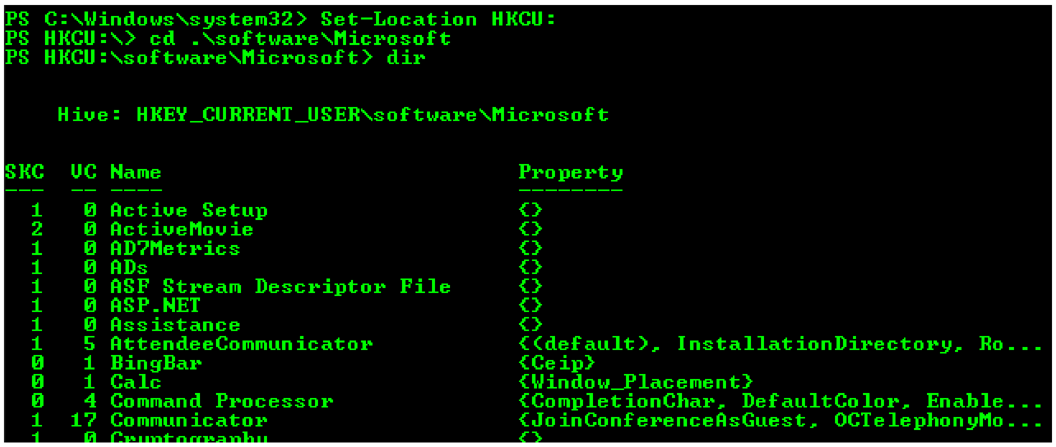
- Note `HKCU` & `HKLM`

**066 PSDrive is an interesting component within PowerShell. You use PSDrive to navigate the registry like a file system. This is kind of neat. You use Get-PSDrive. It'll show you a list of the current drives and the aliases in this particular PowerShell session, and you'll notice the HKEY_CURRENT_USER and HKEY_LOCAL_MACHINE right there--right here, the HKCU and the HKLM are right there.

PowerShell PSDrive -2

Set your root for navigation to the HKEY_CURRENT_USER with

- Set-Location HKCU:
- Navigate using traditional commands like cd, dir



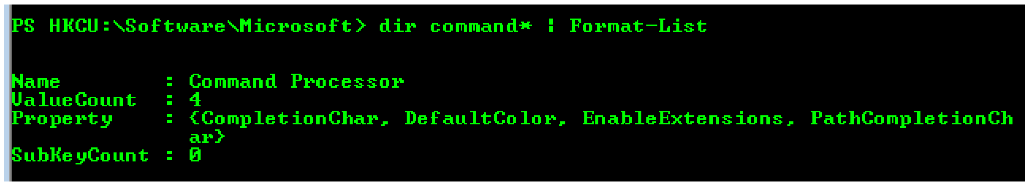
```
PS C:\Windows\system32> Set-Location HKCU:
PS HKCU:\> cd .\software\Microsoft
PS HKCU:\software\Microsoft> dir


    Hive: HKEY_CURRENT_USER\software\Microsoft


SKC  VC Name                           Property
---  -- ----                           --------
  1   0 Active Setup                   {}
  2   0 ActiveMovie                    {}
  1   0 AD7Metrics                     {}
  1   0 ADs                            {}
  1   0 ASF Stream Descriptor File     {}
  1   0 ASP.NET                        {}
  1   0 Assistance                     {}
  1   5 AttendeeCommunicator           {(default), InstallationDirectory, Ro...
  0   1 BingBar                        {Ceip}
  0   1 Calc                           {Window_Placement}
  0   4 Command Processor              {CompletionChar, DefaultColor, Enable...
  1  17 Communicator                   {JoinConferenceAsGuest, OCTelephonyMo...
  1   0 Cryptography                   {}
```

**067 And then once you set your location to HKCU or HKLM, you can actually access that area by using some traditional commands like change directory and then dir, or directory, to show some of the pieces within there. Normally it takes a significant amount-- well, not significant-- but using regedit and certain other things, it's kind of harder to get access to the registry, whereas it's kind of interesting that this makes it pretty simple to use something like change directory, cd, and then a dir to show a listing of information that's inside the registry.

PowerShell PSDrive -3

Cannot view all the properties in a regular `dir`, let's find out more about `Command Processor.`

- `Dir command* | format-list`
- Will show all the properties for any items matching the command* criteria



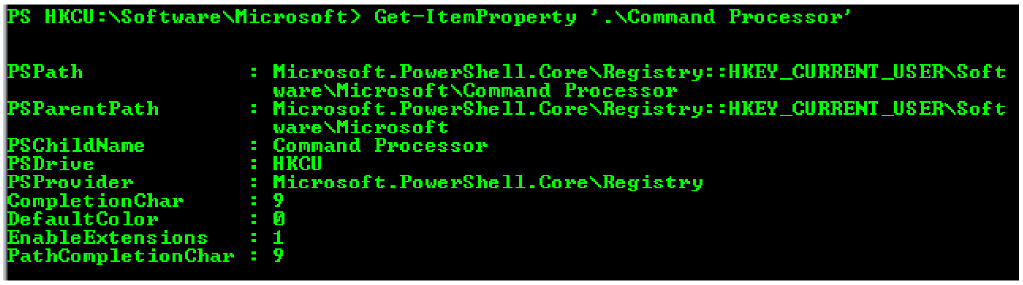
```
PS HKCU:\Software\Microsoft> dir command* | Format-List

Name        : Command Processor
ValueCount  : 4
Property    : {CompletionChar, DefaultColor, EnableExtensions, PathCompletionCh
              ar}
SubKeyCount : 0
```

**068 So you can't necessarily view all the properties like a regular dir, but you can take the dir command star and you can pipe it to a format list and it'll show you properties matching command star criteria. In this case, it'll be the command processor. Then you got your properties there.

PowerShell PSDrive -4

Now to view the value of one of the properties

- `Get-ItemProperty '.\Command Processor'`



```
PS HKCU:\Software\Microsoft> Get-ItemProperty '.\Command Processor'

PSPath               : Microsoft.PowerShell.Core\Registry::HKEY_CURRENT_USER\Soft
                       ware\Microsoft\Command Processor
PSParentPath         : Microsoft.PowerShell.Core\Registry::HKEY_CURRENT_USER\Soft
                       ware\Microsoft
PSChildName          : Command Processor
PSDrive              : HKCU
PSProvider           : Microsoft.PowerShell.Core\Registry
CompletionChar       : 9
DefaultColor         : 0
EnableExtensions     : 1
PathCompletionChar   : 9
```

Changing `CompletionChar` to `"0"` will disable tab completion at the command line.

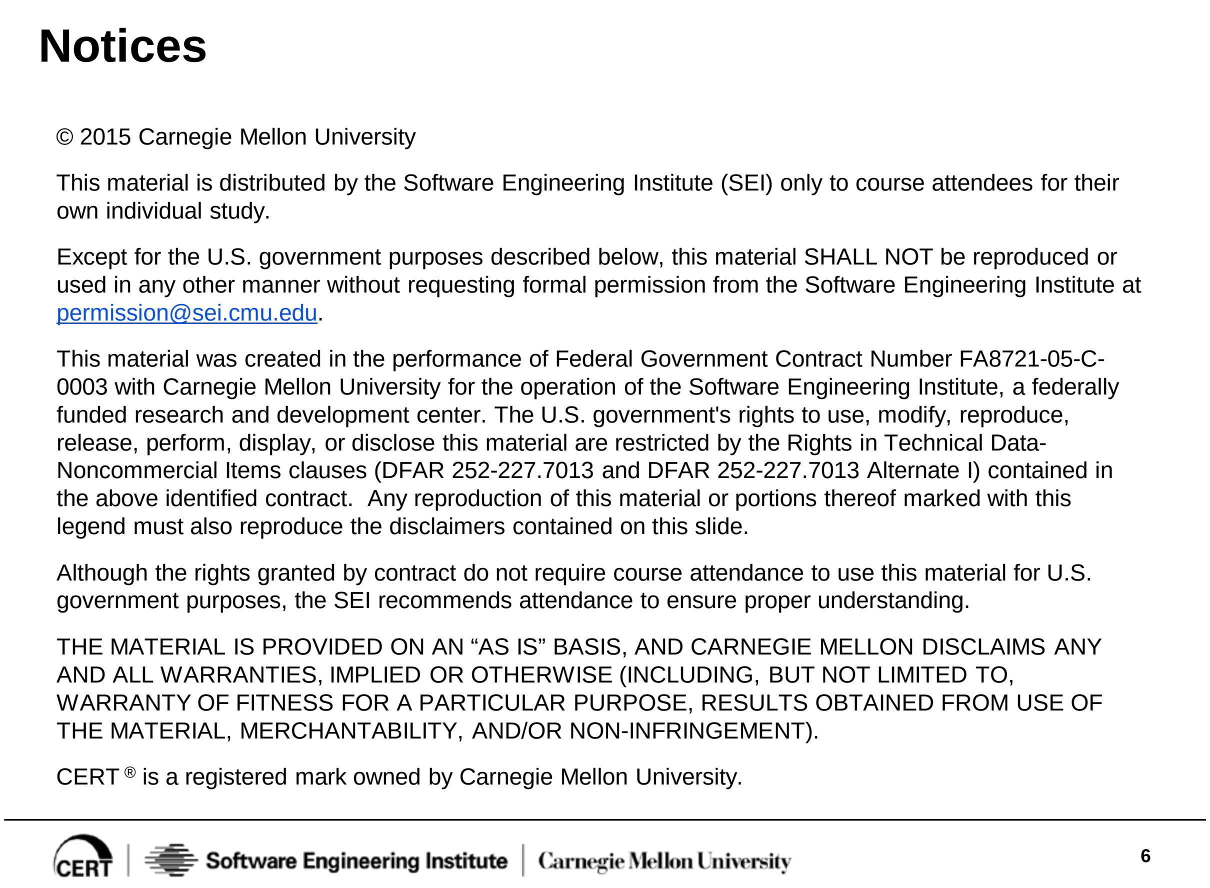
- `Set-ItemProperty '.\Command Processor' CompletionChar "0"`

**069 So you can use Get-ItemProperty, and then .\Command Processor, and it'll give you this information.

So if you generally don't do a lot of administration, lot of dealing with registries, this is kind of obscure information. But just understanding that PowerShell is strong enough and powerful enough to reach in and get this information for you. So should you need it in the future, this is how you could possibly use this, and you saw how you can use this in a batch file so you're able to get at that and launch it through a batch file or some other scripting language.

I'm not sure why you would necessarily want to do this, but you can actually stop the tab completion by changing the CompletionChar to zero, and it'll stop doing that. So perhaps if it's annoying or if it has a bunch of words-- or a word that's always starting with the same few letters or whatever and it always gives you the wrong option, perhaps you would want to turn that off. So it does allow you to turn off the tab completion.

Notices



Notices