



PowerShell Scripting

Cybersecurity
Windows Administration and Hardening Day 2



Class Objectives

By the end of today's class, you will be able to:



Use basic PowerShell cmdlets to navigate Windows and manage directories and files.



Use PowerShell pipelines to retrieve Windows system event logs.



Combine various shell-scripting concepts such as cmdlets, parameters, piping, conditions, and importing files with data structures.



**While many
IT professionals prefer
Mac OS and Linux,
Windows is still the
leader for desktop operating
systems.**

Let's Review

Last class, we learned how to use CMD to execute many Windows sysadmin tasks.



How to audit processes with Task Manager.



Using CMD to create files.



Creating a report with `wmic` in the command line.



Auditing unwanted startup applications and services.



Enumerating local users, groups, and current local password policies.



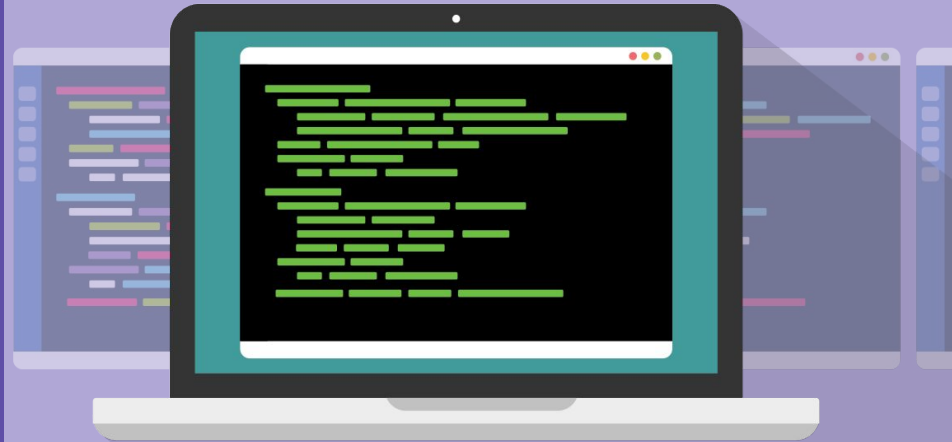
Creating new regular and administrative users and setting local password policies.



Scheduling tasks using Task Scheduler.

While CMD is short, simple and easy to learn, it isn't designed for complex operations and procedures.

On the other hand, PowerShell was designed as a powerful language used to execute, automate, and customize the most demanding and difficult tasks.



Powershell in system administration

PowerShell can be used to manage everything in a Microsoft enterprise environment, including:

Windows Server: Microsoft's central server for domain and networking services.

Windows 10: Microsoft's personal and professional computer operating system. The ubiquitous operating system used by most people and within many large enterprise environments.



Powershell in system administration

PowerShell can be used to manage everything in a Microsoft enterprise environment, including:

Office365: Microsoft's line of subscription office services, included Microsoft Word, Excel and Powerpoint.

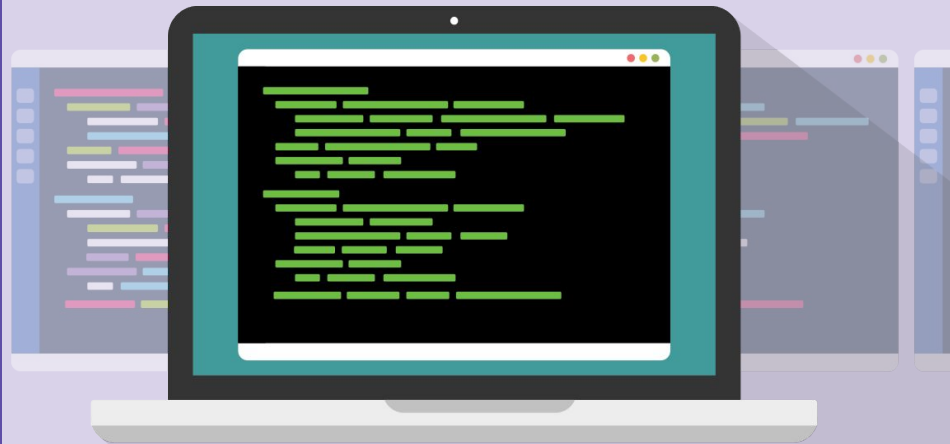
Azure: Microsoft's cloud services, offering a wide variety of cloud services, such as virtual private clouds and cloud based VMs.



Powershell in Cyber Security

Defensive security: PowerShell can be used to manage and audit logs. There are many commands for interacting with Windows Event logs. We will be looking at these later.

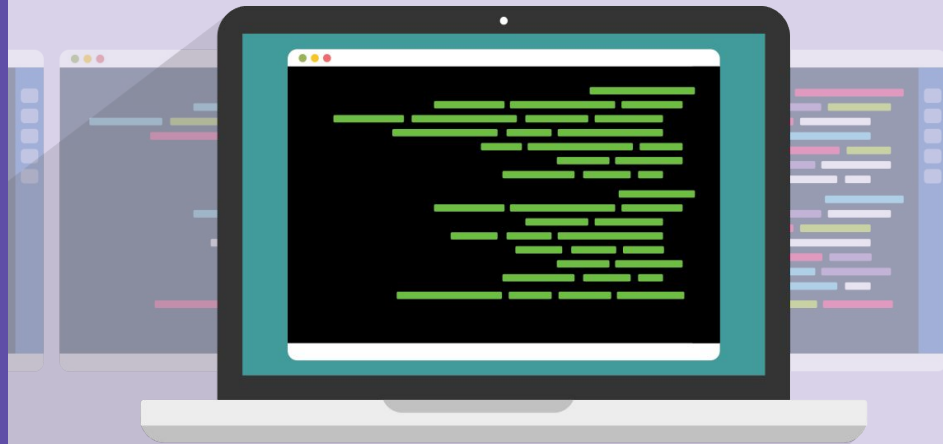
- PowerShell can also be used to harden the security on Windows hosts and servers.
- There are many modules and scripts that extend PowerShell's powerful functionality to enforce cybersecurity policy.



Powershell in Cyber Security

Offensive security: PowerShell is often used as a "living off the land" tool, meaning it is a tool that exists on the target's computer that can be leveraged by attackers.

- Once a system is breached, Powershell is often used to retrieve a wide array of information within a network, such as user and server names.
- It can also be used to access and maintain access on other networked machines, if they are not properly secured.



So, What is PowerShell?

PowerShell is a powerful scripting language that lets us locally and remotely manage Microsoft's line of products.

➡ Since Microsoft enterprise products are the most widely used by organizations, it is critical that system administrators and security professionals know PowerShell.

➡ PowerShell can be used to lock down and harden enterprise networks, leveraged by offensive security professionals, and exploited by malicious actors.



PowerShell vs. CMD

CMD's functionality is limited.

CMD output is only available in simple text format.

- For unsupported file formats, we need to edit the output with meticulous character replacing.

CMD command flags can be ambiguous, confusing, and specific to each command:

- Examples of different `/s` flags:
 - `shutdown /s` shuts down a computer.
 - `freekdisk /s` specifies the name of an IP or remote computer to check disk space.
- Examples of different `/d` flags.
 - `shutdown /d` specifies a reason for shut down.
 - `freedisk /d` specifies which disk drive to check.

PowerShell vs. CMD

PowerShell provides clearly defined, universal parameters for commands.



Based on the language used in the following commands, what do you think they do?

```
Stop-Computer -Confirm
```

```
Stop-Computer -Force
```

PowerShell vs. CMD

PowerShell provides clearly defined, universal parameters for commands.



Based on the language used in the following commands, what do you think they do?

Stop-Computer -Confirm

Shuts down the machine with a confirmation prompt verifying that you want to shut it down.

Stop-Computer -Force

Immediately shuts down the machine.

PowerShell vs. CMD

We can use PowerShell Piping for operations that CMD doesn't support. To find the file sizes of all files in C:\Windows, we can use a complex batch file consisting of the following:

```
@echo off
set size=0
for /r %%x in (System\*) do set /a size+=%%~zx
echo %size% Bytes
```

PowerShell can do this with a simple pipe (|):

```
dir C:\Windows\System -Recurse | Measure-Object -Sum Length
```

- `dir C:\Windows\System -Recurse`: Grabs all the current directory and subdirectory contents.
- `Measure-Object -Sum Length`: The output is piped into this command, which measures files.

Objects are a very important PowerShell concept. We'll look at them next.

What Are Objects?

“Object” is Microsoft’s name for every component in a system that PowerShell recognizes and interacts with.

If we run `ls C:\Windows`:

- All the files and directories in C:\Windows are processed by PowerShell. Each is an object, with its own properties.
- `C:\Documents\Recipes\Guacamole.doc`
 - `Guacamole.doc` is the `file.name` property of the file (object).

What Are Objects?

Understanding everything as objects with properties allows us to use more specific commands to target the results we want.

For example: We can use a pipe to retrieve only objects containing the word “system”:

```
ls C:\Windows | Where-Object {$_.name -like "*system*"}
```

- Lists contents of C:\Windows in which the .name property contains “system.”
- `$_`: Indicates the previous object, referring to C:\Windows.

We'll cover syntax in more depth later.

More PowerShell Benefits



We can confirm we're using the right commands with PowerShell's extensive internal documentation system.



PowerShell uses **aliases** to mimic Unix commands, like **ls** and **cat**.



Despite being a Microsoft product, PowerShell is open source and available on GitHub. By contrast, there's no source code available for CMD, and writing tools are limited to batch scripts.

PowerShell Commands:

cmdlet	Function	Equivalent Command
Set-Location	Changes to specified directory	cd
Get-ChildItem	Returns current directory contents	ls, dir
New-Item	Makes new directory	mkdir
Remove-Item	Deletes file or directory	rm, rmdir
Get-Location	Retrieves path to current directory	pwd
Get-Content	Returns file content	cat, type
Copy-Item	Copies a file from one location to another	cp
Move-Item	Moves item from one location to another	mv
Write-Output	Prints output	echo
Get-Alias	Shows aliases from the current session	alias
Get-Help	Retrieves information about command	man
Get-Process	Retrieves processes running on machine	ps
Stop-Process	Stops specific process	kill
Get-Service	Retrieves list of services	service --status-all

Verbs-Nouns

Consider the following scenario:

- User **Alex** left the company.
- We want to remove their user account from the system, but we want to keep the reports they were working on.
- We need to move the reports files from their user Desktop directory to a directory outside of the user.
- Along the way, we'll show some other useful Powershell commands.

We'll use the following commands:

Set-Location, Move-Item, Get-ChildItem, New-Item, Remove-Item





Instructor Demonstration

Verbs-Nouns


Parameters

Now, we'll use parameters to customize the commands.

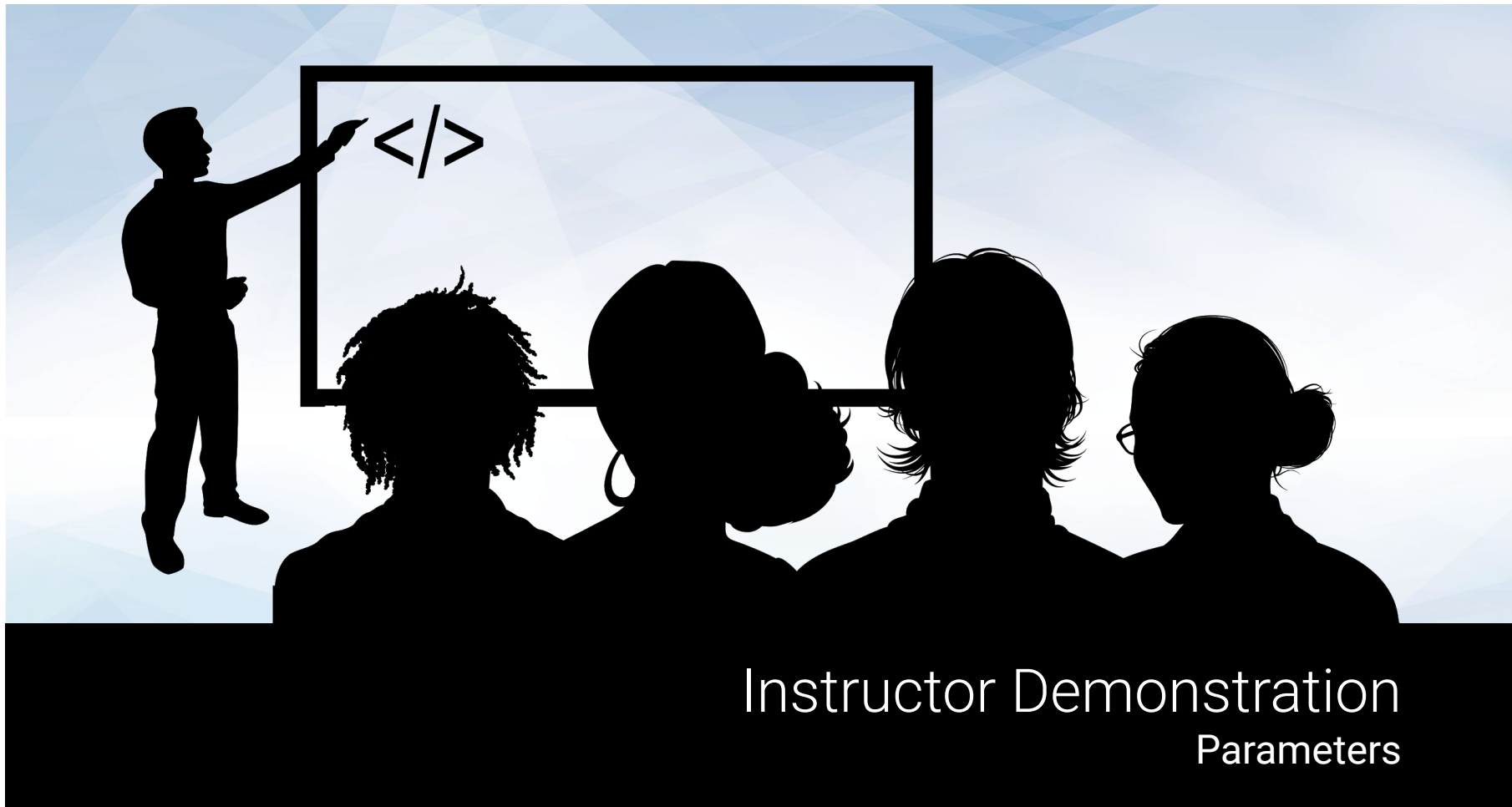
Instead of using **New-Item** to create another file, we can add parameters to the **New-Item** command and create a directory, with a specific name and location:

```
New-Item -Path "C:\\" -Name "Logs" -ItemType "Directory"
```

- **Path**: Parameter specifying the location of this new directory.
- **-Name**: Parameter specifying the directory's name.
- **-ItemType**: Parameter specifying the type of item we want to create.
If we don't specify **"Directory"**, it will default to a file.



Let's
take a look
at some more
parameters...



Instructor Demonstration

Parameters



Activity: Moving and Creating Directories

In this activity, you will work as a junior sysadmin tasked with vetting a process to create Windows Event logs.

First we need to create the appropriate directories to store our information. We'll run PowerShell commands to create, rename, and move items.

Suggested Time:
10 Minutes





Time's Up! Let's Review.

Activity Review: Moving and Creating Directories

To compete this activity, we needed to:

01

Move the **contracts** directory to **C:**.

02

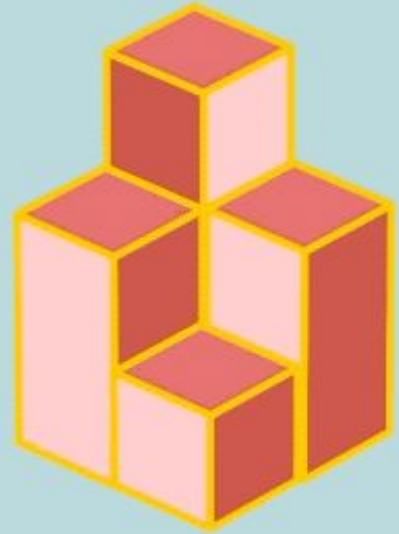
Create **Logs** and **Scripts** directories in **C:**.

03

Rename the **reports** directory as **Reports**.

Generating Windows Event Log Files with Parameters and Pipelines

Now, we will continue building on parameters by chaining commands with pipelines.





Pipeline Demo Setup

In the following demo, we're continuing our role as a junior sysadmin.

Our CIO asked us to retrieve multiple types of logs from our Windows 10 machine and save them as json files in our newly created C:\Logs directory. They will later be imported to a Splunk SIEM for analysis.

Use your cheat sheet to help follow along.



Instructor Demonstration

PowerShell Pipeline

Piping Logs to JavaScript Object Notation with ConvertTo-Json

Now that we've used parameters to get the logs we needed, we will output them into a file that can be later used by log analysis applications.

This is where pipelines come in.





Instructor Demonstration

Piping Logs to json



Activity: Generating Windows Event Log Files

In this activity, you will create and save log files to **C:/Logs**.

Suggested Time:
10 Minutes





Time's Up! Let's Review.

Activity Review: Generating Windows Event Log Files

To compete this activity, we needed to:

01

Execute cmdlets with a variety of parameters and parameter values.

02

Pipe cmdlets together to transform the PowerShell output.

03

Query event logs.

04

Select the most recent logs.

05

Output the logs to a **json** file.



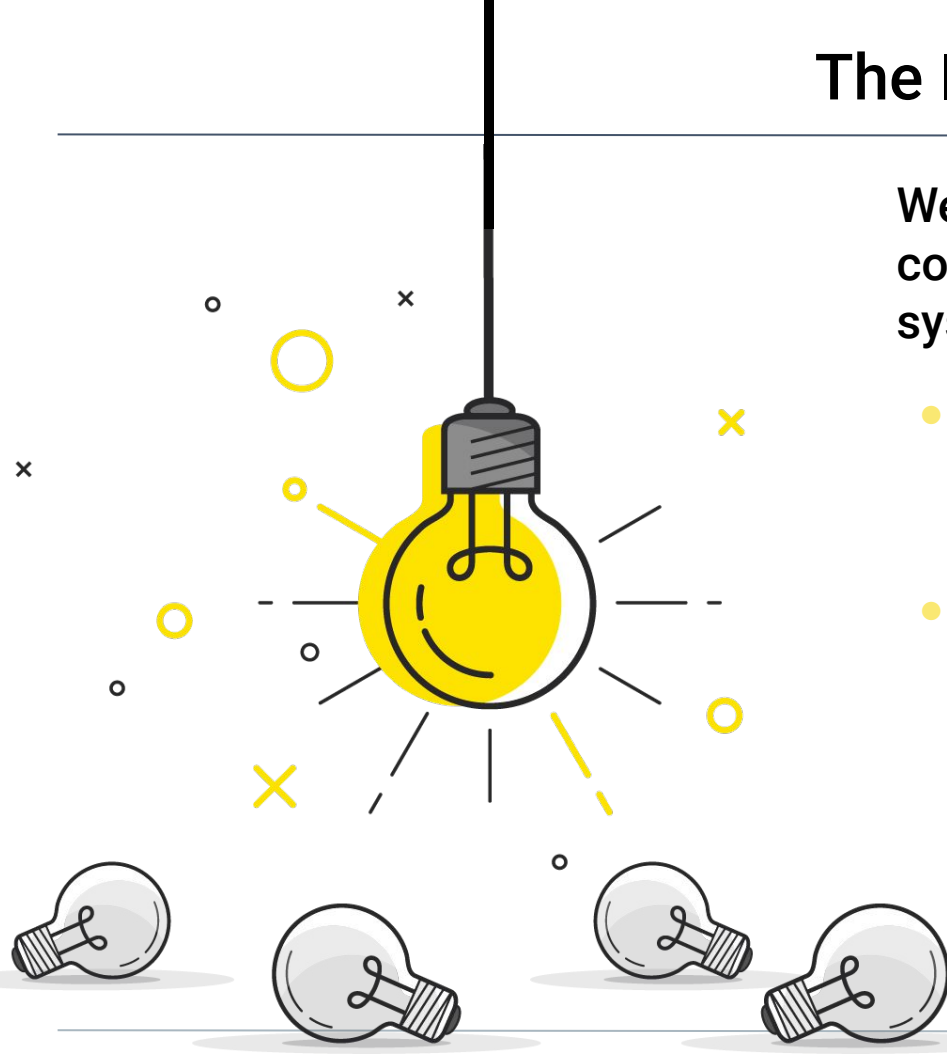
Break

Scripting with PowerShell

The Importance of Scripting (Again)

We've emphasized the importance and convenience of scripting in our past sysadmin units.

- Scripts allow sysadmins and security professionals to automate and execute basic to advanced procedures and operations.
- Scripts can be used for everything from setting basic firewall rules to standing up entire cloud virtual machine environments with networking, storage, and users.

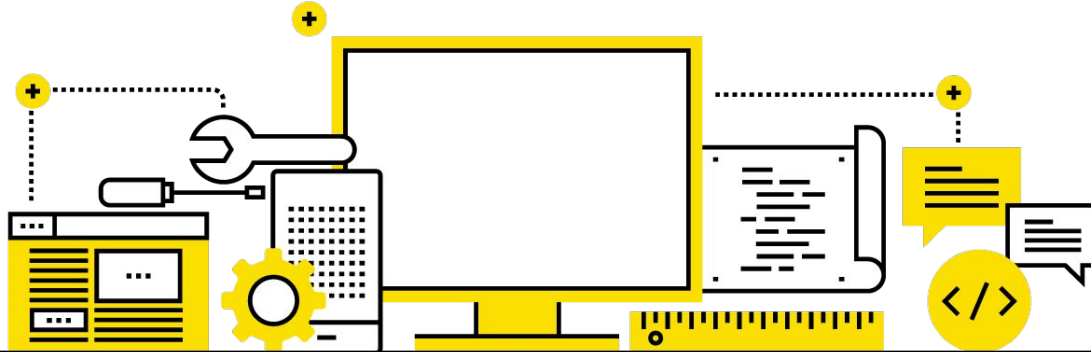


Scripting with PowerShell

Like Linux, PowerShell allows us to script many commands in sequence:

For example, suppose you need to set up Windows workstations for users in the accounting department. You could create a script do the following, in order:

- i. Pull sensitive accounting data and files from a file server to a specified directory.
- ii. Download **AppLocker**, a program for limiting and controlling access to files for certain users and groups.
- iii. Deploy application control policies for **AppLocker** to restrict user access to the directory so only people in the accounting group can access it.



Scripting Demo Scenario

In this demonstration, we will create a script in VS Code to remove *Skype*.

Windows 10 has been notoriously known for implementing a lot of settings and preinstalled applications that have been considered bloat and potential larger attack vectors.

- Some of these settings include **telemetry tracking** and **advertising IDs** while some of the default installed applications include *Skype*.
- We want to remove these applications to reduce the attack surface area for this workstation. Instead of trusting our users to not use these apps, we're going to remove the possibility.





Instructor Demonstration

Powershell Scripts and Removing Skype

We've just deleted a useless app from our machine. While we could remove the Windows Store apps one by one, it would be more time efficient to create a script that will loop through a **list** of the apps and uninstall them all at once.



CSV Files

More specifically, we can loop through **Comma-Separated Values** (CSV) files.

- CSVs are plain text files that contain simply structured data (**fields**) separated by commas.
- The top line of a CSV file contains the **header** — the row that describes each field.

Sysadmins and security professionals will use CSV files containing lists of items they need to parse through.

- A system administrator may use a CSV file to maintain a list of employee email addresses and usernames.
- A penetration tester might have a list of IP addresses and corresponding domain and subdomain names to use in a test..

CSV File

```
appxpkg,name,description
"Microsoft.ZuneMusic","Zune","Microsoft's Zune Music Player"
"Microsoft.Music.Preview","Music Preview", "Microsoft's Music Preview"
"Microsoft.XboxGameCallableUI","Xbox Gaming GUI", "Microsoft's Xbox Overlay"
[CSV contents truncated]
```




appxpkg	name	description
Microsoft.ZuneMusic	Zune	Microsoft's Zune Music Player
Microsoft.Music.Preview	Music Preview	Microsoft's Music Preview
Microsoft.XboxIdentityProvider	Xbox ID Provider	Microsoft's Xbox Live Account Management

foreach Loops

We can use **foreach** loops to iterate through these files' items.

- The **foreach** loop in PowerShell is similar to the **for** loop in Linux, but it is mainly used for looping through files or read-only structured data that you need to loop through.

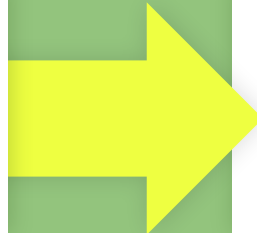


appxpkg	name	description
Microsoft.ZuneMusic	Zune	Microsoft's Zune Music Player
Microsoft.Music.Preview	Music Preview	Microsoft's Music Preview
Microsoft.XboxIdentityProvider	Xbox ID Provider	Microsoft's Xbox Live Account Management

`foreach` Loops and CSV Files

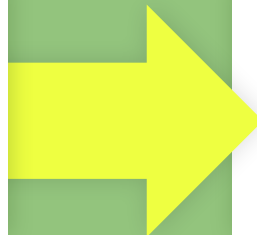
Sysadmins and security professionals will use CSV and `foreach` loops together:

A system administrator may use a CSV file to maintain a list of employee email addresses and usernames.



They can use a `foreach` loop to loop through each item and change the passwords.

A penetration tester might have a list of IP addresses and corresponding domain and subdomain names to use in a test.



They can use a `foreach` loop to try out each password with a known username.



In this demo, we will be using the Import-Csv PowerShell cmdlet to source the CSV file demo.csv to run with a demo PowerShell script.

If the CSV file is constructed with the standard header and record setup, Import-Csv will prepare the fields to be called by PowerShell.



Instructor Demonstration Using Import-Csv with a foreach Loop



Our demo.csv file has multiple rows containing multiple attributes, such as apppkg, name, and description. Sometimes we only want to loop through the Windows Store application names or descriptions.

We can append .name to the \$line variable. The script will return the matching field or attribute entries instead of entire lines. In this case, it returns the application name.



Instructor Demonstration

Importing a CSV File and Looping Through Field Attributes



Activity: Removing Windows Bloat with PowerShell Scripts

In this activity, you'll use PowerShell and a CSV file to create a script that removes unwanted applications.

Suggested Time:
15 Minutes





Time's Up! Let's Review.

Activity Review: Removing Windows Bloat with PowerShell Scripts

To compete this activity, we needed to:

01

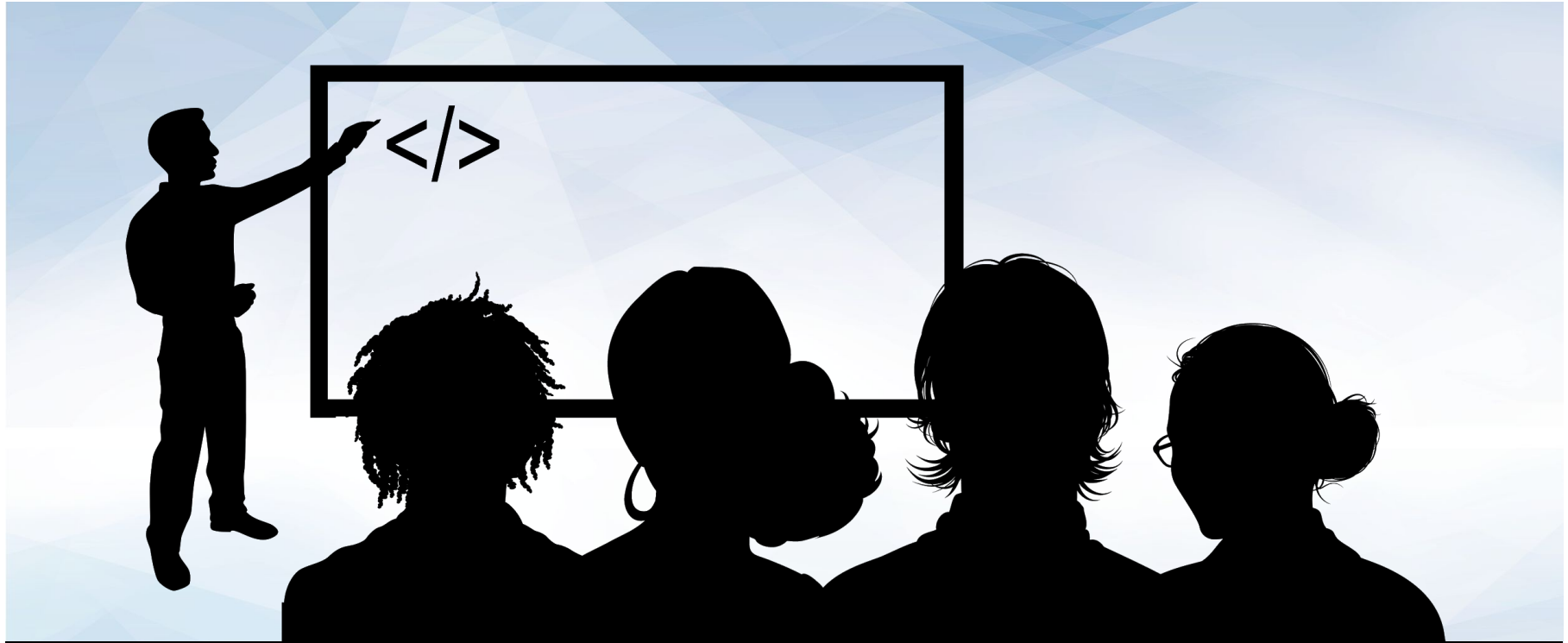
Use the cmdlet **Import-Csv** to import a CSV file as an object into PowerShell.

02

Create a **foreach** condition that loops through each line, for the application name field, in the CSV file.

03

On every loop, execute a two-part cmdlet pipe that retrieves each application object by name, and sends it to the **Remove-AppxPackage -Verbose** cmdlet to uninstall it.



Instructor Demonstration

Windows Remoting



Activity: PowerShell Remoting

In this activity, you will construct a script to send the **Get-WinEvent** security log **.json** files to the Windows Server.

Suggested Time:
15 Minutes





Time's Up! Let's Review.

Activity Review: PowerShell Remoting

To compete this activity, we needed to:

01

Enable PowerShell remoting to trusted hosts. In this case, the Windows Server IP address.

02

Use **Enter-PSSession** to remotely access the Windows Server and set up a **C:\Logs** directory.

03

Create a script that does the following:

- Sets up a variable to set up the **New-PSSession** cmdlet with **-ComputerName** and **-Credential** parameters.
- Adds a **Copy-Item** cmdlet to copy the **RecentSecurityLogs.json** file made in the previous activity to the **C:\Logs** directory on the remote host, with the following parameters: **-Destination** and **-ToSession**.
- Uses **Remove-PSSession \$Session** at the end of the script to end our remote session (since these are persistent sessions).

Setting Up for Day 3

Next class, we will be setting up and working with an Active Directory server.





Setting Up

Next class, we will be setting up and working with an Active Directory server. In order to get it set up we need to:

- Run a script in the RDP host machine that will set up Hyper-V so that our two VMs can communicate with each other.
- During the next lesson we will set up the machines so that they have static IP addresses.
- After that, we will run a script inside the Windows Server VM that will set up Active Directory for us.

Hyper-V PowerShell Commands to Set Up a NAT Switch

The following script will set up a virtual Hyper-V switch, configure it as a NAT network, and then add the two Windows virtual machines to it.

Execute the following script in an administrative PowerShell session within the RDP host lab machine:

```
New-VMSwitch -SwitchName "NATSwitch" -SwitchType Internal

New-NetIPAddress -IPAddress 192.168.0.1 -PrefixLength 24 -InterfaceAlias "vEthernet (NATSwitch)"

New-NetNAT -Name "NATNetwork" -InternalIPInterfaceAddressPrefix 192.168.0.0/24

Get-VM "Windows Server" | Get-VMNetworkAdapter | Connect-VMNetworkAdapter -SwitchName "NATSwitch"

Get-VM "Windows 10" | Get-VMNetworkAdapter | Connect-VMNetworkAdapter -SwitchName "NATSwitch"
```

Stage the Active Directory Set Up Script

Next, download the following script into your Windows Server VM.

Do NOT download this script into the RDP host machine. You do not need to run it now.

If you're unsure how to do this, double-click Windows Server in your Hyper-V Manager. Within that VM, open a browser window. Download the following file.

```
#  
# Windows PowerShell script for AD DS Deployment  
#  
Install-WindowsFeature -name AD-Domain-Services -IncludeManagementTools  
Import-Module ADDSDeployment  
Install-ADDSForest `  
-CreateDnsDelegation:$false `  
-DatabasePath "C:\Windows\NTDS" `  
-DomainMode "WinThreshold" `  
-DomainName "GOODCORP.NET" `  
-DomainNetbiosName "GOODCORP" `  
-ForestMode "WinThreshold" `  
-InstallDns:$true `  
-LogPath "C:\Windows\NTDS" `  
-NoRebootOnCompletion:$false `  
-SysvolPath "C:\Windows\SYSVOL" `  
-Force:$true
```

Stage the Active Directory Set Up Script

In our next lesson,
we will use this
script to setup an
Active Directory
Domain Controller.

```
#  
# Windows PowerShell script for AD DS Deployment  
#  
Install-WindowsFeature -name AD-Domain-Services -IncludeManagementTools  
Import-Module ADDSDeployment  
Install-ADDSForest `  
-CreateDnsDelegation:$false `  
-DatabasePath "C:\Windows\NTDS" `  
-DomainMode "WinThreshold" `  
-DomainName "GOODCORP.NET" `  
-DomainNetbiosName "GOODCORP" `  
-ForestMode "WinThreshold" `  
-InstallDns:$true `  
-LogPath "C:\Windows\NTDS" `  
-NoRebootOnCompletion:$false `  
-SysvolPath "C:\Windows\SYSVOL" `  
-Force:$true
```

IMPORTANT:

Make sure to turn off both your VMs and the host machine at the end of class.



Time's Up! Let's Review.

Any Questions?