POWERSHELL FUNDAMENTALS …

TLO KNOWLEDGE AND SKILLS

Conditions:

- Given a classroom, applicable references, and a practical exercise, the Cyber Mission Force student will demonstrate understanding of PowerShell fundamentals.

Knowledge:

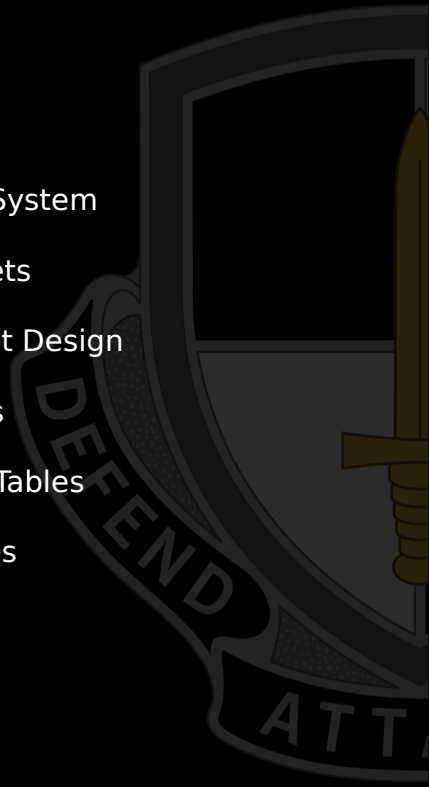
- Identify the structure of PowerShell command line.
- Understand PowerShell’s basic functions and purposes.

Skills:

- Working knowledge of how to use basic commands and aliases in PowerShell.
- Understanding of PowerShell tables, arrays, and how to set up profiles.

OBJECTIVES

- Identify Windows Powershell and its Functions
- Describe PowerShell Structure
- Describe Cmdlet Composition
- Describe Common Usage of PowerShell
- Define Powershell
- Identify Powershell Purpose
- Describe Powershell Aliases
- Create and use Aliases
- Describe powershell Help System
- Describe Powershell Cmdlets
- Describe Verb-Noun Cmdlet Design
- Describe Powershell Arrays
- Describe Powershell Hash Tables
- Describe Powershell Profiles



WHAT IS POWERSHELL?

PowerShell is a cross-platform automation solution made up of a command-line shell, a scripting language, and a configuration management framework. PowerShell runs on Windows, Linux, and macOS.

As a scripting language, PowerShell is commonly used for automating the management of systems. It is also used to build, test, and deploy solutions, often in CI/CD environments.

In software engineering, CI/CD or CICD is the combined practices of continuous integration and either continuous delivery or continuous deployment. CI/CD bridges the gaps between development and operation activities and teams by enforcing automation in building, testing and deployment of applications.

PowerShell is built on the .NET Common Language Runtime (CLR). All inputs and outputs are .NET objects.

An object in Powershell is simply an enriched data container. This container represents data in an organized structure with properties and functions / methods. The methods are actions that can be taken by an object.

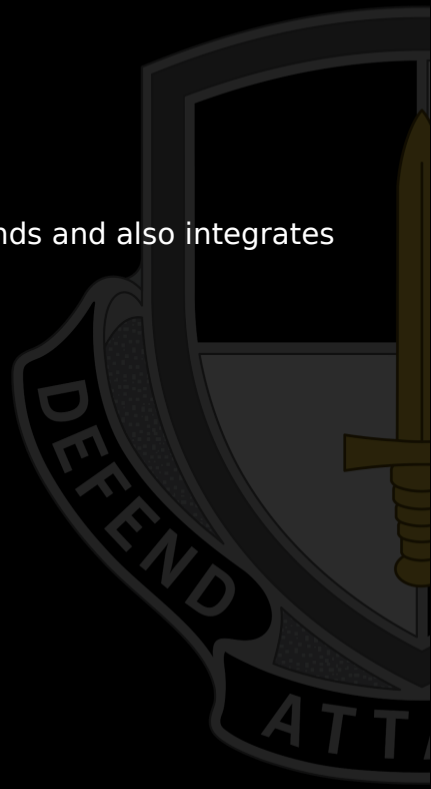
POWERSHELL FEATURES

PowerShell shares some features with traditional shells:

Built-in help system: The help system in PowerShell provides information about commands and also integrates with online help articles.

Pipeline: MA pipeline is used to run many commands sequentially.

Aliases: Aliases are alternate names that can be used to run commands.



HOW IS POWERSHELL DIFFERENT?

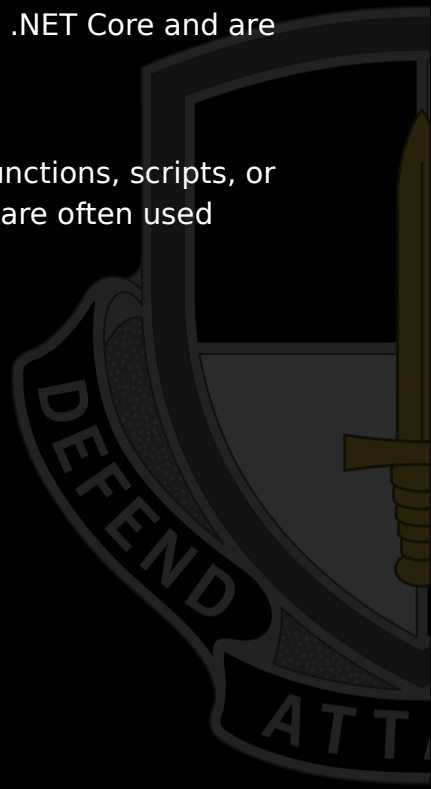
It operates on objects over text. In a command-line shell, you have to run scripts whose output and input might differ. So you end up spending time formatting the output and extracting the data you need. By contrast, in PowerShell you use objects as input and output. That means you spend less time formatting and extracting. Objects retain their properties even as a user manipulates the output they see on their screen or use in pipelines.

Commands in PowerShell are called cmdlets (pronounced commandlets). Unlike many other shell environments, in PowerShell, cmdlets are built on a common runtime rather than separate executables.

HOW IS POWERSHELL DIFFERENT?

Cmdlets typically take object input and return objects. The core cmdlets in PowerShell are built in .NET Core and are open source. You can build your own cmdlets in .NET Core or PowerShell.

It has many types of commands. Commands in PowerShell can be native executables, cmdlets, functions, scripts, or aliases. Every command you run belongs to one of these types. The words command and cmdlet are often used interchangeably.



CHECK ON LEARNING

What are some features PowerShell shares with traditional shells?

What are commands in PowerShell called?

COMMANDLETS

A cmdlet is a lightweight command that is used in the PowerShell environment. The PowerShell runtime invokes these cmdlets within the context of automation scripts that are provided at the command line.

Cmdlets differ from commands in other command-shell environments in the following ways:

Cmdlets are instances of .NET classes; they are not stand-alone executables.

Cmdlets process input objects from the pipeline rather than from streams of text, and cmdlets typically deliver objects as output to the pipeline.

COMMANDLETS

Cmdlets employ a verb/noun naming convention that is designed to make each cmdlet easier to remember and read. As an example, a typical Get-ChildItem command uses the verb Get followed by the noun ChildItem.

When executed through the PowerShell runtime environment, the Get-ChildItem command lists or returns the items in one or more specified locations. If items are in a container, the command gets the items inside the container — child items.

POWERSHELL PIPELINE

Simply put, a pipe | takes the output of the first part of the pipeline, and then uses it as the input of the next part.

```
Steve: C:\Users\student >>>get-service | Select-Object Name

Name
----
AarSvc_4639e
AJRouter
ALG
```

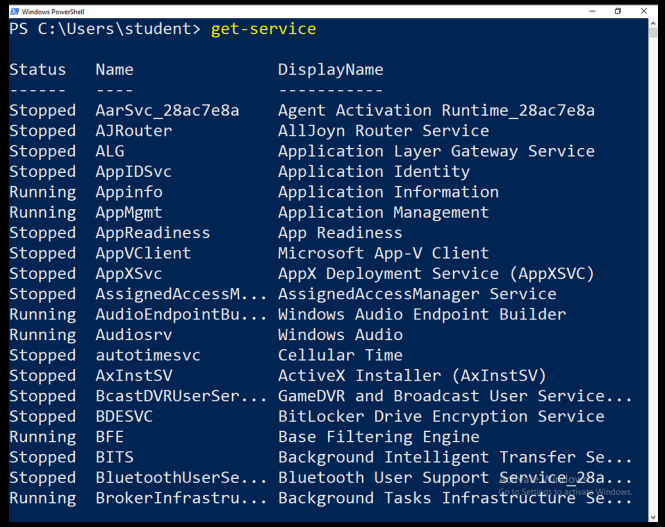
Using the Select-Object cmdlet, how would we display only the DisplayName Property?

PARAMETERS AND ARGUMENTS

Most cmdlets support the use of parameters as part of the input mechanism. Parameters can be added to the cmdlet at the command line or passed to cmdlets through the pipeline as the output from a previous cmdlet.

The arguments or values of each parameter detail the actual input that the cmdlet will accept, how the cmdlet should work and what, if any data the cmdlet outputs.

On the command line, when you run the Get-Service cmdlet, you get a list of services on your machine.



You can further filter these just to show the services that are running:

Get-Service | Where-Object {$_.Status –eq “Running”}

CHECK ON LEARNING

What type of naming convention do cmdlets use?

Which statement is true about how cmdlets differ from commands?

a. Cmdlets are stand-alone executables.

b. Cmdlets are instances of .NET classes

c. both a & b

POWERSHELL HELP SYSTEM

Most shells have some kind of help system in which you can learn more about a command. For example, you can learn what the command does and what parameters it supports.

Powershell Help Cmdlets:

- Get-Help
- Get-Command
- Get-Member

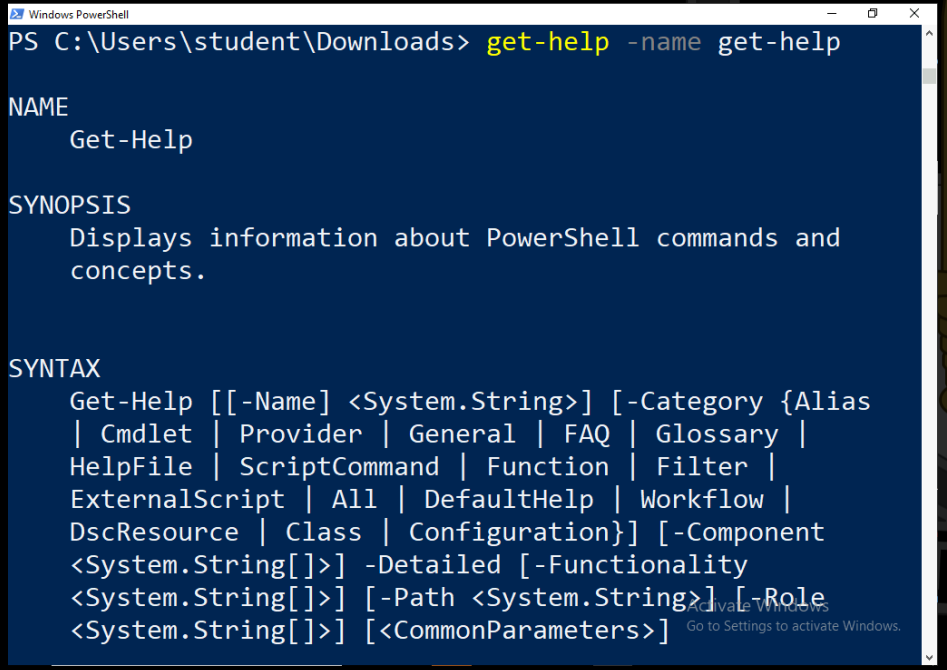
GET-HELP

Get-Help is a multipurpose command. Get-Help helps you learn how to use commands once you find them.

When Get-Help is used to locate commands, it first searches for wildcard matches of command names based on the provided input. If it doesn't find a match, it searches through the help topics themselves, and if no match is found an error is returned.

Get-Help -Name Get-Help

We are calling "Get-Help" on the cmdlet named "Get-Help"



PARAMETERS

Take a moment to run that example on your computer, review the output, and take note of how the information is grouped:

NAME

SYNOPSIS

SYNTAX

DESCRIPTION

RELATED LINKS

REMARKS

As you can see, help topics can contain an enormous amount of information and this isn't even the entire help topic.

PARAMETERS

While not specific to PowerShell, a parameter is a way to provide input to a command. Get-Help has many parameters that can be specified in order to return the entire help topic or a subset of it.

The syntax section of the help topic shown in the previous set of results lists all of the parameters for Get-Help.

At first glance, it appears the same parameters are listed six different times. Each of those different blocks in the syntax section is a parameter set. This means the Get-Help cmdlet has six different parameter sets. If you take a closer look, you'll notice that at least one parameter is different in each of the parameter sets.

PARAMETER SETS

Parameter sets are mutually exclusive. Once a unique parameter that only exists in one of the parameter sets is used, only parameters contained within that parameter set can be used. For example, both the Full and Detailed parameters couldn’t be specified at the same time because they are in different parameter sets.

Each of the following parameters are in different parameter sets:

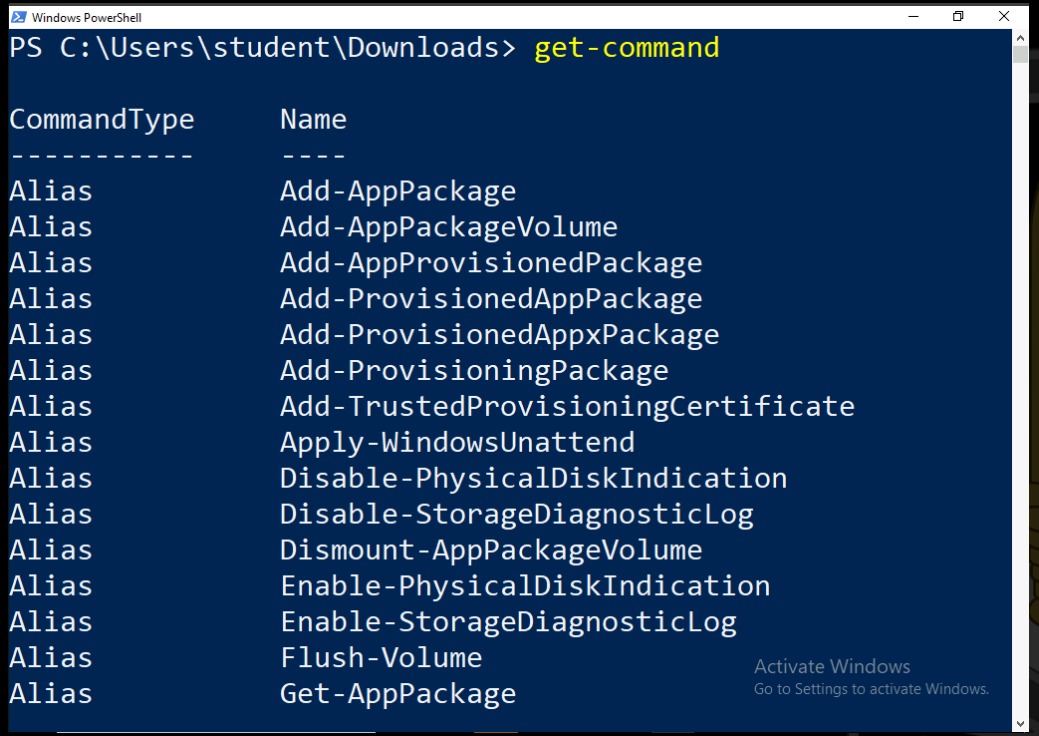
- Full
- Detailed
- Examples
- Online
- Parameter
- ShowWindow

GET-COMMAND

The Get-Command cmdlet gets all commands that are installed on the computer, including cmdlets, aliases, functions, filters, scripts, and applications.

This is helpful when we want to get a list of all the commands that are available to us in the current powershell session.

Syntax: Get-Command



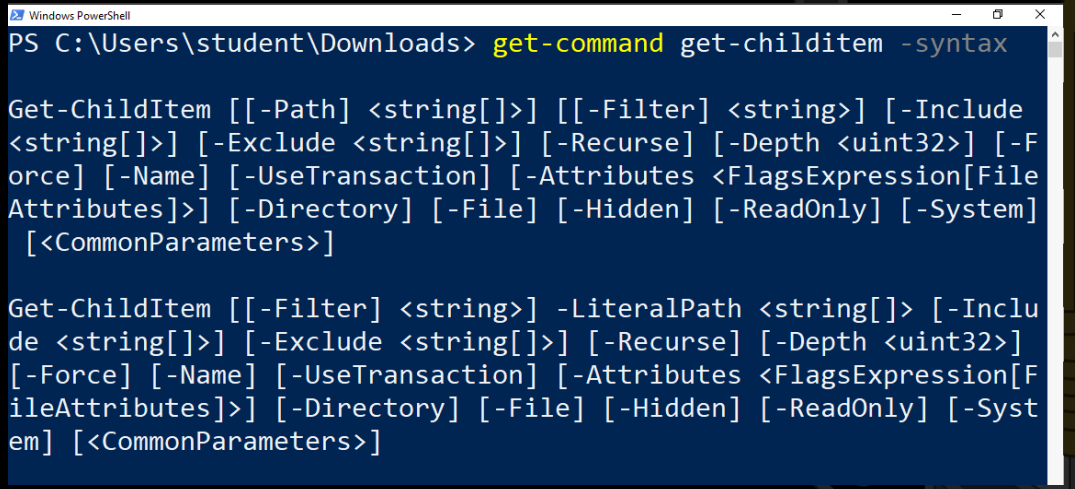
GET-CHILDITEM

This command grabs the parameter sets of the Get-ChildItem command

Syntax: Get-Command -Name Get-Childitem -Syntax

We can replace Get-ChildItem with any command we want to get the syntax for.

We are calling "Get-command" on the cmdlet named "Get-Childitem" to see what the accepted syntax is.



WILDCARD *

A wildcard character “*”, simply matches any and all string filters.

Syntax: Get-Command Get*

We are calling "Get-command" and filtering on command that starts with Get

GET-MEMBER

The Get-Member cmdlet gets the members, the properties and methods, of objects.

Syntax: get-process | get-member

Takes Get-Process and pipes it into Get-Member so that we can see all the methods and properties of that object. Think of the methods as “things we can do with these objects,” and properties as “things these objects are.”

CHECK ON LEARNING

What does the get-member cmdlet do?

True or False: Parameter sets are not mutually exclusive.

POWERSHELL ALIASES

An alias is an alternate name for a cmdlet, function, executable file, including scripts. PowerShell includes a set of built-in aliases. You can add your own aliases to the current session and to your PowerShell profile.

PowerShell supports the use of common aliases such as cls (clear the screen) and ls (list the files). PowerShell includes built-in aliases that are available in each PowerShell session. So new users can use their knowledge of other frameworks and don’t necessarily have to remember the PowerShell name for familiar commands.

We can add new aliases for our own session of Powershell.

To create an alias, use the cmdlets Set-Alias or New-Alias.

Syntax: Set-Alias -Name list -Value Get-Childitem

POWERSHELL ALIASES

We can verify our alias was created using the Get-Alias cmdlet.

Syntax: Get-Alias -Name list

We have now created an alias called list. This will function EXACTLY the same as Get-ChildItem.

Using the Definition parameter, you can find all the aliases of a command. Let’s find all the current aliases of the Get-ChildItem cmdlet.

CHECK ON LEARNING

What is the full command to create an alias?

True or False: The get-alias cmdlet can verify an alias was created

VARIABLES

A variable is a unit of memory in which values are stored. In PowerShell, variables are represented by text strings that begin with a dollar sign ($), such as $a, $process, or $my_var.

Variable names are not case-sensitive in Powershell, and can include spaces and special characters.

USER CREATED VARIABLES

User-created variables are created and maintained by the user. By default, the variables that you create at the PowerShell command line exist only while the PowerShell window is open. When the PowerShell windows is closed, the variables are deleted.

To save a variable, add it to your PowerShell profile. You can also create variables in scripts with global, script, or local scope.

```
#User-created variable
$my_var = 1
echo $my_var

$my_var += 1
echo $my_var
```

AUTOMATIC VARIABLES

Automatic variables store the state of PowerShell. These variables are created by PowerShell, and PowerShell changes their values as required to maintain their accuracy. Users can’t change the value of these variables. For example, the $PSHOME variable stores the path to the PowerShell installation directory.

```
#Automatic variable
echo $PSVersionTable
echo $PSHome
```

PREFERENCE VARIABLES

Preference variables store user preferences for PowerShell. These variables are created by PowerShell and are populated with default values. Users can change the values of these variables. For example, the $MaximumHistoryCount variable determines the maximum number of entries in the session history.

```
#Preference variable
echo $MaximumAliasCount
echo $MaximumHistoryCount
```

CHECK ON LEARNING

List some of the different types of variables discussed in Powershell

True or False: Users can change preference variables

ARRAYS

An array is a data structure that is designed to store a collection of items. The items can be the same type or different types. An array is organized by zero-based index.

To create and initialize an array, assign multiple values to a variable.

The values stored in the array are delimited with a comma and separated from the variable name by the assignment operator (=).

```
$A = 1,5,10,15,10,"squirrel"
#In this example we are taking the values 22,5,10,8,12,9 and 80 and storing them into the array named $A

echo $A
```

ARRAYS

This data is now associated in memory with the array named $A. Anytime we want to access this information we can interact with the array it is stored in.

```
#We can access the value at a specific position by refering to its index.

echo $A[1]
```

The value 5 is stored at index 1.

ARRAY SUB EXPRESSION

The array sub-expression operator creates an array from the statements inside it. Whatever the statement inside the operator produces, the operator will place it in an array. Even if there is zero or one object.

Sub Expression Syntax

```
@( commands )
```

Sub Expression Syntax

```
$services = @(get-service | where {$_.Status -eq "Running"} | Select-Object Name )
```

Now we have an array named $services that is a collection of all the service names with a status of "Running."

ARRAY SUB EXPRESSION

Now we have an array named $services that is a collection of all the service names with a status of "Running"

We can now interact with our service names as if they were just a regular array.

```
echo $services[12]
```

We've taken output from a commandlet and have organized it into an array for further data processing or analyzing.

One of the properties of an array is the length. This is simply how many objects are in an array. We can access this value using the .length method of an array.

HASH TABLES

A hash table, also known as a dictionary or associative array, is a compact data structure that stores one or more key/value pairs.

For example, a hash table might contain a series of IP addresses and computer names, where the IP addresses are the keys and the computer names are the values, or vice versa.

HASH TABLES

To create a hash table, follow this syntax:

```
$phonebook = @{ Bob = "706-123-4567"; Alice = "803-123-4567"; Steve = "555-123-4567"}
```

We now have an array where keys are associated with their corresponding values.

Unlike a regular array hash tables use key based indexing. We cannot refer to values in their numerical position. We have to reference the specific key in order to access the value.

```
echo $phonebook["Bob"]
echo $phonebook["Alice"]
```

CHECK ON LEARNING

What is an array?

What are the values stored in an array delimited with?

POWERSHELL FEATURES

You can create a PowerShell profile to customize your environment and to add session-specific elements to every PowerShell session that you start.

A PowerShell profile is a script that runs when PowerShell starts. You can use the profile as a logon script to customize the environment.

You can add commands, aliases, functions, variables, snap-ins, modules, and PowerShell drives. You can also add other session-specific elements to your profile so they are available in every session without having to import or re-create them.

POWERSHELL PROFILE FILES

- PowerShell supports several profiles for users and host programs. However, it does not create the profiles for you.

Description	Path
All Users, All Hosts	**Windows** $PSHOME\Profile.ps1 **Linux** /usr/local/microsoft/powershell/7/profile.ps1 **macOS** /usr/local/microsoft/powershell/7/profile.ps1
All Users, Current Host	**Windows** $PSHOME\Microsoft.PowerShell_profile.ps1 **Linux** /usr/local/microsoft/powershell/7/Microsoft.Powershell_profile.ps1 **macOS** /usr/local/microsoft/powershell/7/Microsoft.Powershell_profile.ps1
Current User, All Hosts	**Windows** $Home\Documents\PowerShell\Profile.ps1 **Linux** ~/.config/powershell/profile.ps1 **macOS** ~/.config/powershell/profile.ps1
Current user, Current Host	**Windows** $Home\Documents\PowerShell\Microsoft.PowerShell_profile.ps1 **Linux** ~/.config/powershell/Microsoft.Powershell_profile.ps1 **macOS** ~/.config/powershell/Microsoft.Powershell_profile.ps1

EDITING A PROFILE

You can open any PowerShell profile in a text editor, such as Notepad.

Here we are taking advantage of the automatic variable $PROFILE

```
notepad $PROFILE

#To specify a specific profile.
notepad $PROFILE.AllUsersAllHosts
```

Test the path of the $PROFILE to see if it exists

```
Test-Path $profile
```

EDITING A PROFILE

It may show false because the file doesn’t exist, so let’s create it.

```
New-Item –Path $Profile –Type File –Force
```

Now if we test it again it should show us true. Type $PROFILE to see what the default path to the file we just created is. Navigate to it and open it up. In here, the profile file, we can make any changes that we want to happen everytime we open the console.

EDITING A PROFILE

We are going to add the following to our profile:

```
function prompt{
        "CYBERMAN: $($executionContext.SessionState.Path.CurrentLocation) $('= ')"
}
set-alias -name list -value get-childitem
```

To apply the changes, save the profile file, and then restart Powershell

CHECK ON LEARNING

True or False: A powershell profile is a script that runs when PowerShell starts?

List two of the PowerShell profiles