PowerShell

1. Cmdlets: Verb – Noun

Examples:

set-location c:\

get-childitem 🡪 dir

clear-host 🡪 cls

get-alias

Unix commands && Dos commands also work:

ls 🡪 list screen

md jason

mkdir jeffrey

gal 🡪 alias for [ get-alias ]

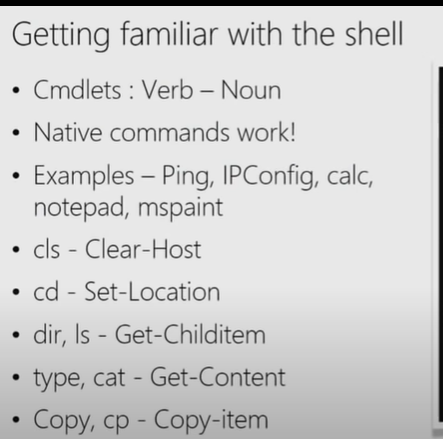
Windows native commands:

notepad

calc

mspaint

ipconfig 🡪 ping dc



gal sa\*

saps 🡪 Start-Process

savm 🡪 Start-VM

spsv 🡪 Stop-Service

gsv 🡪 Get-Service

get-alias -Definition get-process

2 in result/ output

2nd module: Using help

update-help -Force To get the additional help package

man command / help command 🡪 complete help

get-help 🡪 normal help

get-help \*service\* 🡪 Lists all servers that beings or ends with service

get-help g\*service\*

get-verb

get-verb | measure 🡪 98

get-help get-service -Detailed

get-help get-service -Examples

get-help get-service -full

get-help get-service -Online

get-help get-service -ShowWindow 🡪 Settings

Using parameters:

get-service -Name bits, bfe

get-service -DisplayName Bit\*

get-eventlog -LogName System -Newest 3

get-eventlog -LogName System -Newest 3 -EntryType error -ComputerName dc, s1, s2

CLS ; HELP about\_Eventlogs

get-help -category Provider

get-help Certificate

Module 3: The pipeline: Getting connected and Extending the shell

* What’s the pipeline and what does it do?
* Exporting / Importing CSV
* Exporting / Importing XML
* Other files and printers
* Displaying information in a GUI
* Making a webpage of information
* Cmdlets that kill

get-service -name bits | start-service -PassThru

get-service | stop-service 🡪 Very dangerous,

get-service | export-csv -Path C:\service.csv

notepad c:\service.csv 🡪 not the correct file to view output

import-csv c:\service.csv 🡪 outputs on console in format

get-process | Export-clixml -Path c:\good.xml

Building a pipeline, checking if we can make software on two computers exactly the same.

get-service | convertTo-html -Property name, status | out-file c:\test.htm

C:\test.htm 🡪 opens the webpage created

Dangerous actions Precautions:

get-service | stop-service -whatif

get-service | stop-service -confirm

Module 04) Extending the Shell

commandlets ship with the things that need to be managed.

we do this through modules

mmc 🡪 Opens snap-ins

Get-Module -ListAvailable

get-adcomputer -Filter \*

Module 5: Object for the Admin

get-service b\*

doesn’t output text, it results in object

get-process | where handles -gt 900 | sort handles

Objects have properties and methods (Handles – process id)

properties – columns

get-service -name bits | get-member

Shows me all the properties

get-service | Select -Property name, status

get-childitem (Files and folders in my C drive)

get-childitem | Select -Property name,length | sort -Property length -Descending

When you need to know more about an object just use [ get-service | gm ] (get-member)

get-eventlog -LogName system -newest 5 | select -property eventid, timewritten, message | sort -Property timewritten | convertTo-html | out-file c:\error.htm

c:\error.htm

XML FILE

x.PLAY.ACT.SCENE.SPEECH | group speaker | sort count

Bruce’s Book – powershell in action Designing the language syntax

1. get-service | where {$\_.status -eq “Running”}

get-help \*comparison 🡪 why -eq ?

1. This can be written better

get-service | where {$PSItem.status -eq “Running”}

or

get-service | where {$PSItem.status -eq “Running” -and $\_.name -like “b\*”}

get-stuff | Sort | where -somestuff | out-file

get-stuff | where -somestuff | Sort | out-file

2nd one is better. Why ?

powershell.org

gps | where handles -ge 1000

**Learn powershell in a month of lunches , Chapter 9**

Module 6) Deeper into PowerShell

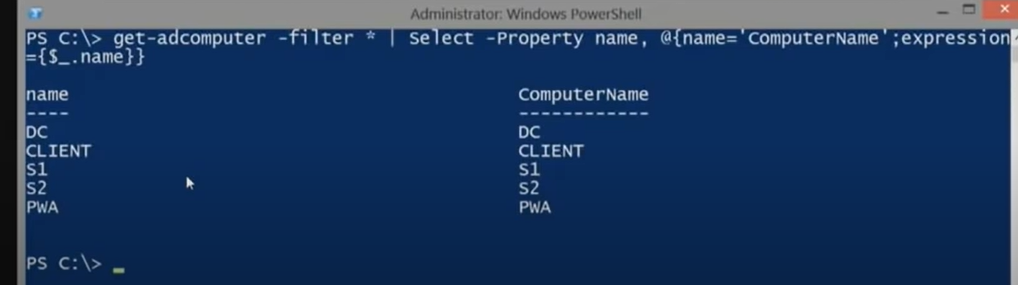
How the pipeline really works – The 4 step solution

1. ByValue
2. ByProperty Name
3. What if my property doesn’t match – Customize it!
4. The parenthetical – when all else fails.

get-AdComputer -filter \* | Select -Property name, @{n = ‘ComputerName’ ; e={$\_.name}}

get-AdComputer -filter \* | Select -Property, @{n = ‘ComputerName’ ; e={$\_.name}} |

get-service -name bits



1st way – byValue, 2nd way – byPropertyName, 3rd way – When PropertyName does not match

4th way When all else fails—

\\\\\\

Get-wmiobject -class win32\_bios -ComputerName (Get-AdComputer -filter \* | Select -ExpandProperty name)

in v3 🡪 ….(Get-AdComputer -filter \* ).name

\\\\\\\

wmiobject – new cmdlet -- ciminstance

Another way to write that

Get-ADComputer -filter \* | get-wmiobject win32\_bios -computerName {$\_.name}

Next module))

invoke-command -comp s1 {get-service -name bits}

invoke-command {get-service -name bits} same as get-service -name bits

The difference is that the first command is deserialized (check with gm, get-members cmdlet). Once, sent to other computers, it becomes deserialized, so when you check it under gm, it shows many methods which means we can manipulate it and do a lot of things unlike the latter.

Module 8)) Automation in Scale – Remoting

$servers | foreach{start iexplore <http://$_>} 🡪 shows empty

notepad C:\default.htm

MVA and PowerShell Rocks ! 🡪 editing the notepad

$servers | foreach{copy-item c:\default.htm -Destination [\\$\_\c$\inetpub\wwwroot](file:///\\$_\c$\inetpub\wwwroot)}

$s = New-PSSession -ComputerName dc 🡪 stored session in this

Import-PSSession -session $s -Module ActiveDirectory -Prefix remote

get-help \*remoteAD\*

get-remoteADComputer -filter \*

$c = get-command get-process

$c.parameters

$c.parameters[“Name”]

Get-command get-\*Adcomputer



(get-command get-remoteAdcomputer).defintion

* Dynamically generated code

Getting details from other machines, still works if versions do not match.

$s = nsn new session

Import-PSSession $s -CommandName get-process -Prefix wow

You can also do it on your local box not only domain controllers.

Module )) Introducing scripts and toolmaking

**PowerShell ISE**

Get-WmiObject win32\_logicaldisk -filter “DeviceID = ‘c:’”

Get-WmiObject win32\_logicaldisk -filter “DeviceID = ‘c:’” | select freespace

ctrl + space for TAB automcomplete check

A computer screen shot of a blue screen

Description automatically generated

module import I think

.\Diskinfo.ps1 -ComputerName -dc

made a parameter

Diskinfo.ps1

1. param(
2. [string]$ComputerName=’localhost’ 🡪 if you don’t write string, will be object (get-help)
3. $bogus
4. )
5. Get-WmiObject -computername $ComputerName -class win32\_logicaldisk -filter “DeviceID=’c:’” |

I made a parameter bogus, I will use it now

.\Diskinfo.ps1 -bogus

1. [cmdletBinding()]
2. param(

[Parameter(Mandatory=$True)] 🡪affects only next parameter

[string[]]$ComputerName=’localhost’ 🡪 if you don’t write string, will be object (get-help)

$bogus

1. )

A computer screen with white text

Description automatically generated

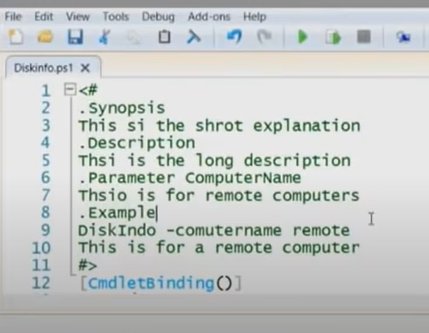
by making the parameter mandatory – ComputerName,

this is how it asks me to enter computerName even If I do not enter it along with .\Diskinfo.ps1

A screen shot of a computer

Description automatically generated

we can also build help for ourselves in the ps1 script



now **get-help .\Diskinfo.ps1** will work

also, **get-help .\Diskinfo.ps1 -full** will also work

You can also ‘CTRL + J’ to get help with

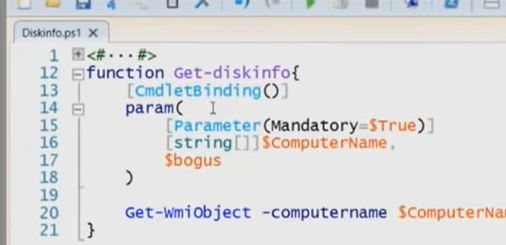
parameterized scripts

.\Diskinfo.ps1 🡪 creates a stack when you run

We can retain it in the stack if you do the following –

.\Diskinfo.ps1 🡪 here the function that I created inside for instance, cannot be used after the command has ran, so the next command is a great way to test things out.

* . .\Diskinfo.ps1
* Get-diskinfo -ComputerName dc 🡪 Get-diskinfo is the function I created in the script

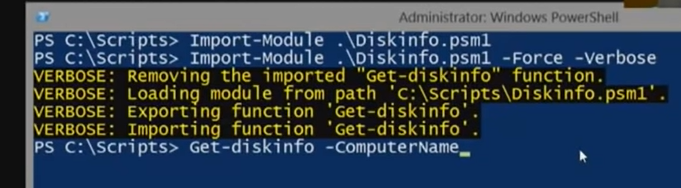


You can also store that into a variable

just use -OutVariable a

$a

gives the output (Can also use, errVariable, warningVariable, etc)



.psm1 instead of .ps1 🡪 you have a module!!

Scripting and Toolmaking

1. Get started scripting
2. PowerShell’s scripting language
3. Simple scripts and functions
4. Advanced functions
5. More on parameters
6. Writing help
7. Error handling
8. Tools that make changes
9. Script and manifest modules

Learn PowerShell Toolmaking in a month of lunches

by Don Jones and Jeffrey Hicks

Microsoft Marketplace course 55038 Windows PowerShell Scripting and ToolMaking

Module 1) Get started scripting

ISE is a third party powershell ISE

Module 2) Powershell scripting language

1. Variables
2. Quotatoin marks
3. Object members and variables
4. Paraenthesis
5. Logical constructs
6. Looping constructs

[String]$MyName = “Jason”

Strongly typed variable ^^

[int]$Oops = “Jason” 🡪Gives an error

Casting 🡪$x = [int]”1”

[int]$x = “1”

$x

1

$x = “test” 🡪 can only store int

[datetime]$d = “10/03/2024”

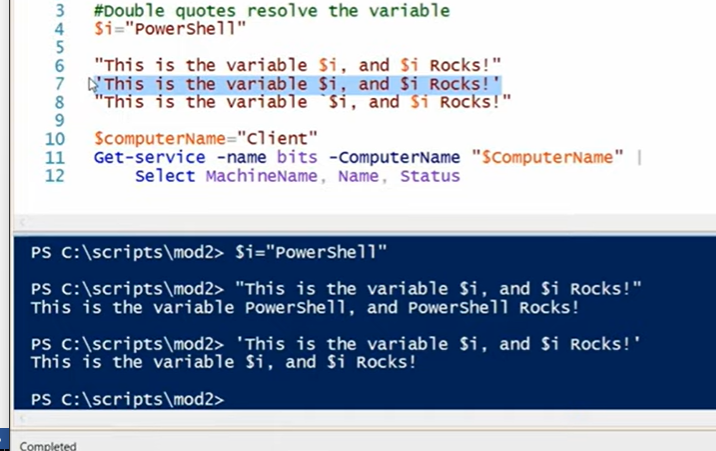
$d.AddDays( -(7\*6))

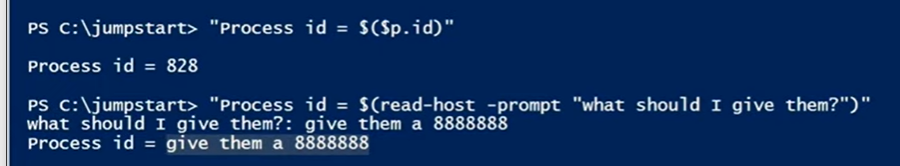
All variables come from .net

MSDN

[Validateset(“a”,”b”,”c”)] [String]$x = “a”

Quotes do matter 🡪





New-Isesnippet -Description “This is a test which shows stuff”…..

Title is \_MYVARTEST

Author: Jeffrey

@’

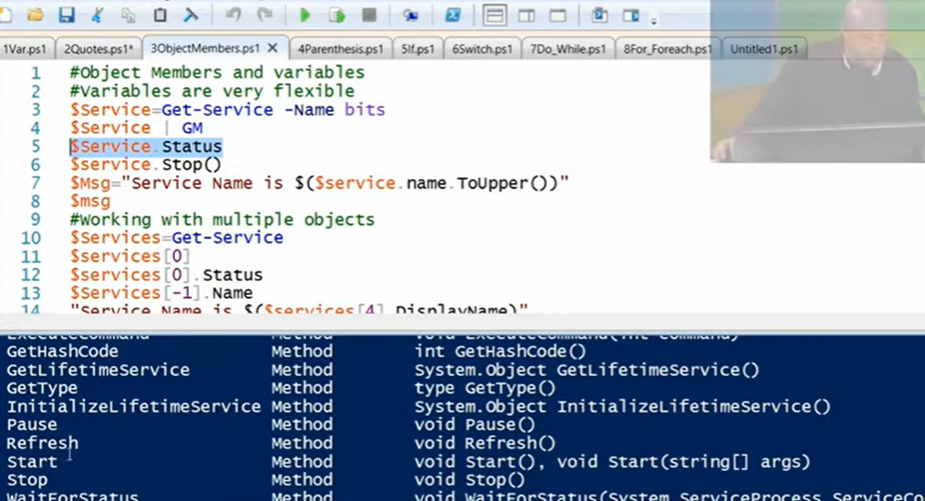
script \_MYVARTEST.. multi line

‘@

open another script, and you can see \_MYVARTEST when you click CTRL + J

3rd example

We are going to put object into a variable



range operator in arrays

1..4

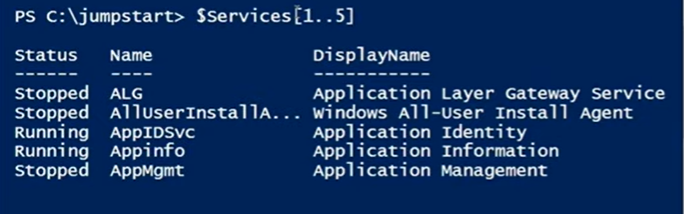
1

2

3

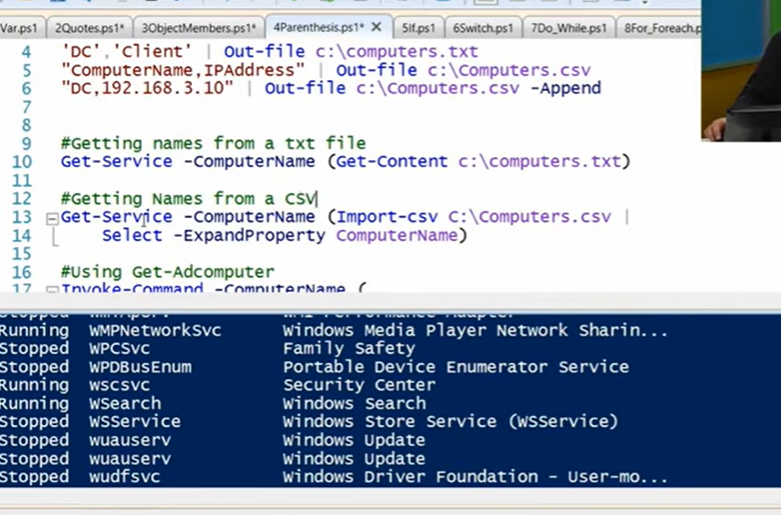
4

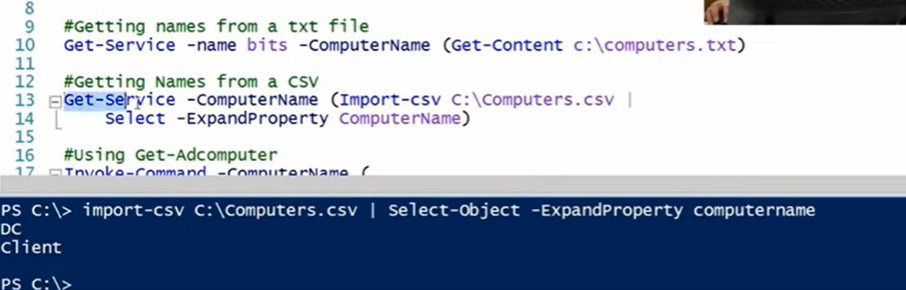
$services[1..5]

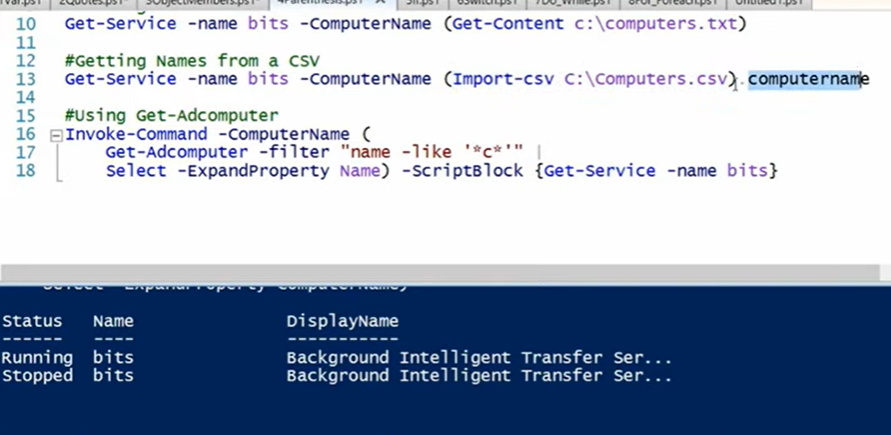


You can also print it in reverse by

$services[5..1]



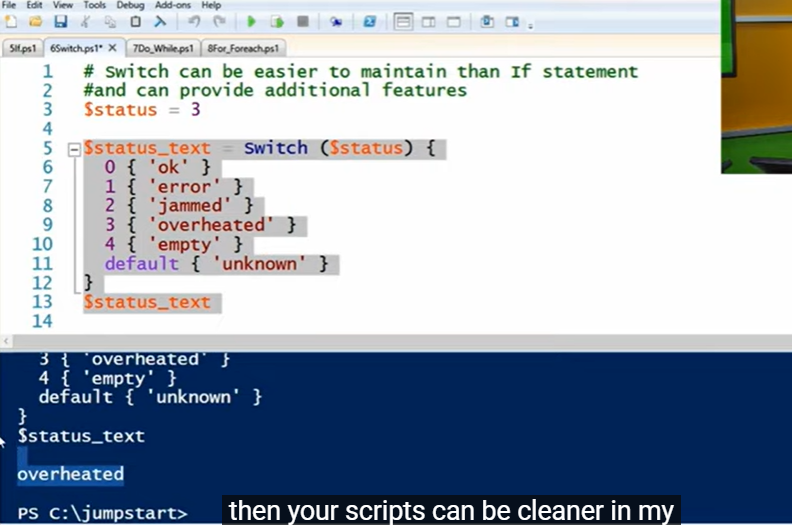




version3 🡪 Using dot operator instead of select and expandProperty

Using-get-Adcomputer

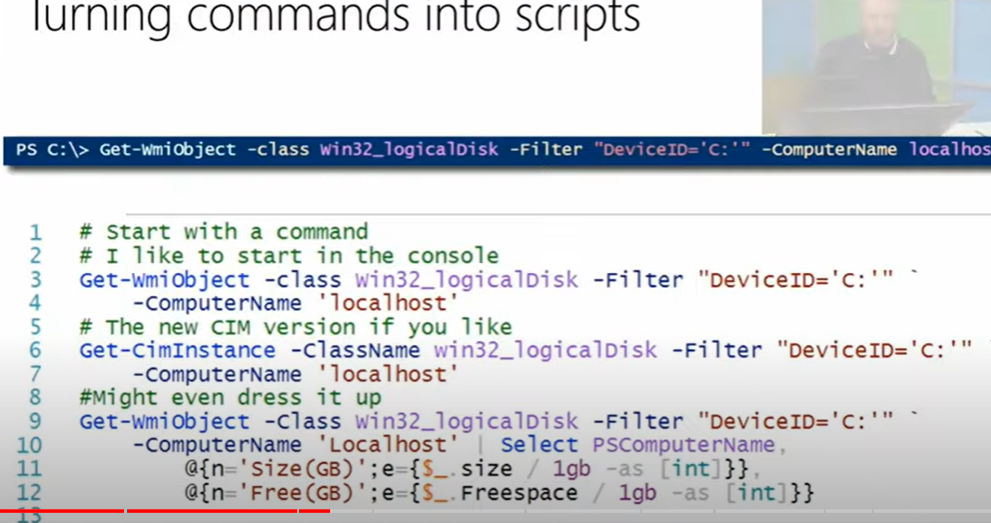
took a very long time to run because it creates a temporary session and then throws that all away by closing the session and result in an output.



Module 3) Simple Scripts and functions

* Turning command into scripts for automation
* Choosing your variables to parameterize
* Adding parameters to your script
* Functions
* Running you script/functions

DEMO 1





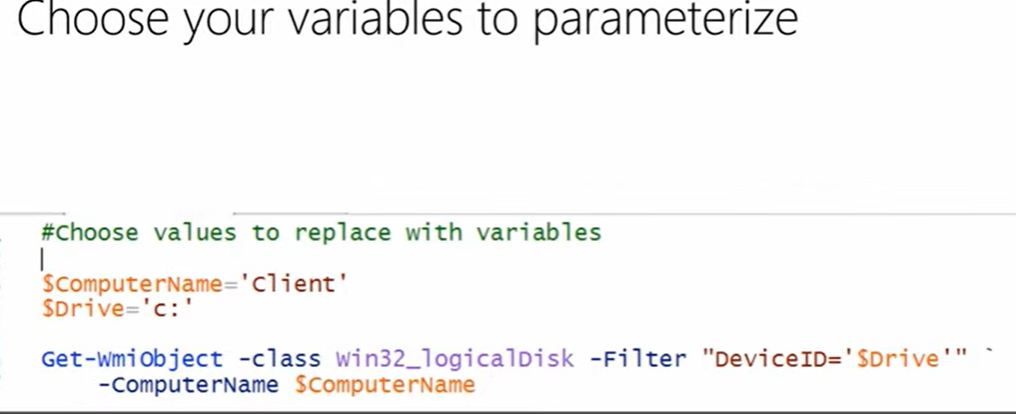
Just by using param, I can now enter the parameters into the console such as 1) Computer Name and 2) Drive

Now DEMO 3 is to parameterize the script

we are using

[CmdletBinding()] – here we are transforming the function as a comdlet

DEMO 2 was to parameterize your variables



function t{

[CmdletBinding()]

param(

[String]….

)

Get-wmiobject -class win32\_bios -Filter ….

}

if you run . .\3Params.ps1

t can be put into console to run function t

Inside the ISE, you do not have to DOT SOURCE it

You have to dot source again if you make changes

A blue background with white text

Description automatically generated

to clean the dot source and everything that’s building up

you can click on CTRL + T = to open a new console

Function Get-Cool {Write-Output “PowerShell Is fun!”}

Function Get-MoreCool {Write-Output “PowerShell….”}

A screenshot of a computer program

Description automatically generated

Module 4)) Advanced functions

1. The purpose to advanced functions
2. An advanced function template
3. Creating and testing parameters
4. Adding your code
5. Writing objects out to pipeline

You can create functions that are similar to cmdlets using PowerShell

get-help \*function\*

A screenshot of a computer program

Description automatically generated