Power Shell

It is task-based Command line & Scripting language. It is designed specially for system administator & power-users to rapidly automate administration of multiple operating systems. It is based on .net framework. It supports case-sentive option.

3 ways through which Powershell can launched:-

→ Command Prompt
→ Powershell Console host
→ Powershell ISE

Cmdlet:- (Command-let):-

→ It is lightweight windows Power-Shell Script that Perform a single function.
→ windows Powershell uses "verb-noun" naming system for names of Cmdlets, where each Cmdlet name consists of standard 'verb' hyphenated with specific ~~noun~~ 'noun'.
'verb' identies action that the Cmdlet Performs
'noun' identifies the entity on which action is performed.

1) retrieve information about the version for window Powershell

→ Get-Host
verb noun.

2) used to retrieve all Commands that are registered in Powershell.

~~Get-Command, Function, Cmdlets~~.

→ Get-Command ~~for~~ Command types, name, modulename info. gives.

3) With the help of -Name Parameter you can Search Specific Cmdlet also.

→ Get-Command -Name Get-Host

if you want to seach only verb & noun Parameter Cmdlets then you can,

→ Get-Command -Verb Get → for showing all 'Get' Cmdlets

→ Get-Command -Noun Host → for showing all 'Host' Cmdlets

4) To retrieve all info. such as syntax, fuction, alias, example and Properties about Specific ~~Co~~ cmdlet.

→ Get-Help

→ Get-Help <Cmdlet-Name> → for help of Particular Cmdlet.

→ Get-Help <Cmdlet-Name> -online → To get online help.

→ Get-Help * → for all list of available help topics. (name, category, module, synopsis)

→ Get-Help Get-Host → for Particular Cmdlet help.

Powershell Parameter!-

Every Cmdlet associated with one & more Parameters always to Perform Certain actions, hence retrieving desired info. Every Cmdlet Comprises Switch & value as shown.

Get-Help -Name Get-Command

-Name → Switch

Get-Command → value

this powershell has three Parameters:

1) optional Parameter
2) mandatory "
3) Positional "

1) The Cmdlet with optional Parameters having it's switch can be executed without specifying square brackets.

→ Get-Help -Name Get-Command

→ Get-Help (Get-Command → here without specifying (-name) switch it can be executes.

But 'verb & noun' it's mandatory to specify.

2) Parameters with no outer square brackets around Parameter (switch & value) indicates it's mandatory Parameter.

It is CmdLet Parameter can't be executed without specifying value to Parameter. these parameters also have optional switches.

→ Get-Eventlog -logname Application

mandatory Parameter, 'value' is mandatory

→ Get-Eventlog Application

3) A Positional Parameter requires that you type of the arguments in relative order.

→ Copy-Item -LiteralPath D:\file1.txt -Destination D:\file2.txt.

if you are not specifying [-LiteralPath & Destination] Parameter then by default it's taking Source & Destination.

Alias :-

Alias is an alternate name that can be created for any windows PowerShell cmdlet. Powershell also has predefined built-in alias for certain cmdlets.

The following cmdlets are used to work with alias,

1) Get-Alias
2) New-Alias
3) Set-Alias
4) Remove-Item

1) It can also return aliases depending all the aliases.

→ Get-Alias → for Alias cmdlets displays

→ Get-Alias -Name w* } Alias shows whatever 'w' names there in front & back.
w

→ Get-Alias -Definition Get-Command → Definition Parameter is used to retrive Alias for Particular cmdlet.

2) Cannot be used to re-assign an existing alias to another cmdlet.

→ New-Alias -Name Commands Get-Command → for Creating new alias / do Get-Command cmdlet Give any name

→ Get-Alias -Definition Get-Command → for showing Alias for Particular cmdlet.

3) ~~Re~~ Reassign an existing alias to a different cmdlet unlike new-alias.

→ Set-Alias -Name Commands Get-Host → by using Set-alias we can allign to another cmdlets. but inbuild Alias we can't allign.

→ Get-Alias -Definition Get-Host → For checking alias to Particular Cmdlet.

→ Set-Alias -Name Host Get-Host → Alias creating & allign to Particular Cmdlet.
↓
Givename

PowerShell variables:-

Variables are objects that can store data. In Powershell variable name starts with '$' character
A variable can be given any name. it can contain a mix of letters, numbers, symbols or even spaces.
we can assign a value to a variable using the Assignment (=) operator.

→ $myname="vasu" → storing variable
↓
$myname → calling variable
↓
vasu → Answer

→ ${"hellow come"}="Devops" → storing variable
↓
${"hellow come"} → calling variable
↓
Devops → Answer

for variable type know:-

→ $myname.GetType()
↓
string

→ ${"hellow come"}.GetType()
↓
string

→ what Properties & methods are available for the variable by using 'Get-member' Cmdlet.

$welcome | Get-Member (or) $MyName | Get-Member

→ we can use one cmdlet output to another variable also,

↳ $Commands = Get-Command

↓

$Commands → All 'Get-Command' output shows here with help of variable

↳ "Hellow $welcome"

↓

Hellow vasu → Answer

Variable Type Casting :- 1) user defined variable.

$a=10 → type = Int32

$b = 20.11 → type = Double

$c = "30" → type = String

↳ $a+$b = 30.11 (Int32 + Double)

↳ $a+$c = 40 (Int32 + String)

↳ $a.GetType() → Int32

↳ $b.GetType() → Double

↳ $c.GetType() → String

↳ [int]$d = "40"

↓

$d.GetType() → Int32

PowerShell Processes the operators in following Precedence order.

negative no.

(), -, *, /, %, "+ or -"

2) Automatic variable :-

↳ $Home → Contains full path of home directory

↳ $PSHome → full path of Installation directory for window PowerShell.

↳ $PSVersionTable → display about version of PowerShell.

↳ $FALSE → Contains false
↳ $True → Contains true

} Instead of using strings (true & false) you can used these variables in Script.

↳ $Null → Contains null & Empty value.

3. enviorment variable :-

↳ these variable have a name & value.

↳ these variable describe the environment in which the program run.

Types :-

↳ user environment variable :- they are specific to your user account & store user specific data.

↳ System environment variable :- they are global & cannot be changed by any user. their values are same for all user accounts.

Read-write Cmdlet:-

* Reading input from the user & Displaying output in Console.

→ $Name = Read-Host "Please enter name"

↓

Reading as variable.

↓

Please enter name: vasu

↓

→ $password = Read-Host "Please enter Password" -AsSecureString

↓

Please enter Password: *****

↓

→ write-Host "My name is $Name"

↓

My name is vasu.

Powershell Scripts:-

A Powershell Script is a plain text file that Contains one or more window Powershell Cmdlets. the extension of Powershell Script is .PS1.

In Powershell we have four different types of execution policies.

1) Restricted (default):- Prevents any script from being run.

2) AllSigned :- Script will run only if they have been Signed by trusted Publisher.

3) RemoteSigned:- Scripts Created locally will run. but those downloaded from internet will not run unless they are digitally Signed by trusted Publisher.

4) Unrestricted:- All Scripts will run.

→ checking current execution policy..

↳ Get-ExecutionPolicy

↓

Restricted (or) RemoteSigned

→ To change execution Policy.

↳ Set-ExecutionPolicy Unrestricted

↓

ExecutionPolicy changed. then you can run all PowerShell Scripts.

Variable Scope:-

In Powershell the scope of variable refers to current environment in which being used. Powershell scopes have a parent-child relationship. Scope of variable is decided based on the place of its declaration.

The following scope types:-

1) Global (default scope).
2) Script
3) Private
4) Numbered

1) Global:-

variable scope with global can be accessed by all the objects within the current Powershell session.

↳ $var="Global"

↓

$var

↓

Global

2) Script :- Can be accessed only within the scripts, can't be accessed by the objects outside the scripts.

↳ $var1 = "Global"

↓

↳ Get-Content scriptScope.ps1

```
$var1
$var1 = "within script"
$var1
```

↓

Global
within script

3) Private Scope :- Can be accessed only within the current scope.

↳ Get-Content PrivateScope.ps1

```
$Private:var1 = "script"
write-Host $var1

function scopeDemo()
{
write-Host $var1
$var2 = "within function"
write-Host "$var2"
}
scopeDemo
```

↓

script
Global
within function.

4) Numbered Scope :-

used to refer the variable with the same name in different scopes. variable are identifying referring it using a name or a number.

↳ Get-Content NumberScope.PS1

```
$num="Numbered Scope"
function funct1
{
    $num="within funct1"
    function funct2
    {
        $num="within funct2"
        Get-variable num -Scope 0
                "       "       "       " 1
                "       "       "       " 2
    }
    funct2
}
funct1
```

↓

Name	Value
num	within funct2
num	within funct1
num	Numbered Scope

Flow Control Statements :-

A flow control statement specifies the block of code to be executed based on condition.

It is classified into :

Selection	Iteration	Branching
* If	* While	* Continue
* While	* Do-while	* Break
	* Do-Until	
	* for	
	* ForEach	

If Statement:-

* In an "if" statement, a condition is tested. if condition is true, a set of statement following 'if' condition are executed.
* if condition is false 'elseif' condition will be tested. if condition is true a set of statement following 'elseif' condition are executed.

↳ $a="powershell" → create "if.ps1" file & enter following code.

```
If ($a -eq "powershell")
{
   "statement is true"
}
ELSE
{
   "statement is false"
}
```

↓

script execute & give result.

→ ./if.ps1

↓

statement is true

Switch Statement:-

It is selection control statement that helps to select a choice from a set of available choices.

Create file, switch.ps1

↓

```
$var="L1043"
Switch ($var)
{
L1043 {"you are in L1043 classRoom"}
L2130 {"    "   L2130   "    "}
GR3   {"    "   GR3     "    "}
```

↓

you are in L1043 classRoom //

While Loop!-

It is used to repeat set of statements as long as the condition is true.

When the condition becomes false, the "while" loop exists.

↳ Get-Content whiledemo.PS1

```
$n=Get-Process | Measure-object | Select-Object -ExpandProperty Count
while ($n -gt 100)
{
    write-Host $n "Processes are running"
    $n-- # Assume we are killing Process to reduce load
}
```

no.of Processes Counts & refer to variable 'n'.

Count

↓

109 Processes are running

:

101 " " "..

DO-while Loop!-

used to repeat set of statements as long as the condition is true. In "do-while" the condition is tested at the end of loop.

↳ Get-Content dowhile.PS1

```
$n=Get-Process | Measure-object | Select-Object -ExpandProperty Count
do
{
    write-Host $n "Processes are running"
    $n--
} while ($n -gt 100).
```

↓

112 Process are running

:

101 " " ".

DO-until Loop:-

It is similar to "do-while loop". But repeats a set of statement as long as condition evaluates to false. when condition becomes true, the loop exists.

↳ Cmt-Content dountil.ps1

```
$n=Get-Process |Measure-object | select-object -expandProperty Count
do
{
  write-Host $n "Process are running"
  $n-- #
} until ($n -lt 100)
```

↓

```
111 Process are running
:
100 Process are running.
```

For Loop:-

Used to execute certain set of statements a specific no. of times.

↳ Processes names redirects to particular file,

Get-Process | select-object name | out-file -filePath testt.txt

↳ Get-Content forloop.ps1

```
$text=Get-Content testt.txt
$pattern="*PowerShell"
For ($i=0 ; $i -lt $text.length ; $i++)
{
if ($text[$i] -like $pattern)
{
"line $i Contains $pattern"
}
}
```

↓

Line 61 Contains *Powershell*

ForEach Loop!-

used to loop through a set of objects & executes an action against each of the object.

↳ Get-Content foreach.ps1

```
$var= Get-Process
ForEach ($a in $var)
{
Write-Host "The Process id of " $a.name "is" $a.id.
}
```

↓

The Process id of aetcrss1 is 4592
" " amsvc is 1880
:
The Process id of chrome is 6484.

Break Statement!-

Breakes the flow of execution work. forces termination of loop. when a "break" is encountred in a loop, the loop terminates immediatly & the execution resumes with the next Command block following loop.

↳ Get-Content break.ps1

```
$var= Get-Process
ForEach ($a in $var)
{
$name= "Powershell"
if ($a.name -eq $name)
{
Write-Host " The Process id of Powershell is" $a.id
break
}
}
```

The Process id of powershell is 13296.

Continue statement:-

It forces the next iteration of the loop take place. Skips the command block b/w 'Continue' statement & end of loop.

↳ Get-Content Continue.PS1

```
$var= Get-Process
ForEach ($a in $var)
{
$name="Powershell"
if ($a.name -ne $name)
{
Continue
}
write-Host "The Process id of Powershell is" $a.id
}
```

The Process id of Powershell is 13296.

Get-Service Cmdlet:-

The Get-Service Cmdlet retrieves information about all the services on local or remote computer.

↳ Get-Service. → for showing services status, name, DisplayName.

→ for retriving the particular Service name,

↳ Get-Service -Name DHCP

↳ Get-Service -Name "*net*" → whatever before & after 'net' words there. Here are services showing here.

→ for retriving the particular DisplayName,

↳ Get-Service -DisplayName "*net*"

Pipelining :- (1) :-

* Powershell allows you to control cmdlets & objects through a technique called Pipelining.
* It is used to execute multiple cmdlet in sequence. As each cmdlet executes, its output is taken as an input to the next cmdlet.
* An object that is pipelined is called as "current pipeline object" & represented by "$_".

Where-Object Cmdlet :-

The where-object cmdlet allows you to limit or filter the object being passed to the pipeline.

↳ Get-Service | where-object {$_.Status -eq "Running"} →
for showing All running services.

↳ Get-Service | where-object {$_.Name -like "n*"} →
for showing services which starting n or N.

↳ Get-Service | where-object {$_.Name -clike "N*"} →
"n*"
only showing services which starting with N or n.

Parameter	works as
-lt	less then
-le	less then equal to
-gt	greater than
-ge	greater then equal to
-eq	Equal
-ne	not equal to
-like	Pattern matching
-clike	Pattern matching with case sensitive

Select-object Cmdlet:-

* Select-object is a filtering Cmdlet that selects specific Properties of an object based on the Parameter specified.

↳ Get-Service | Select-Object -Property Name → only Services 'Names' showing here.

↳ Get-Service | Select-Object -Property Name, Status → only Services 'Names & Status' showing here

↳ Get-Service | Select-Object -First 3 → first three Services showing here.

↳ Get-Service | Select-Object -last 5 → last 5 Services showing here.

Note:- The -ExpandProperty Parameter when used with the Select-object Cmdlet will retrive only the values of the Property specified & doesn't include its Name. In the

Export Data to a file:-

↳ Get-Service | out-file -FilePath services.txt → Get-Service all data storing in services.txt file.

↓

↳ Get-Content Services.txt → Shows Content of file.

↳ Get-Service | Select-Object -last 10 | Export-Csv -Path services.csv → for export to CSV format file.

↓

dir & ls → showing all files

Get-Process:- This Cmdlet retrives information about all the Processes that are currently running on system.

↳ Get-Process
(Name, Id, CPU, VM, WS, PM, Handles shows for Processes).

→ for knowing Particular Process Name, & Id.

~~↳ Get-Service~~

↳ Get-Process -Name Powershell

↳ Get-Process -Id 37876

→ for knowing list of Process with Particular Name starts with,

↳ Get-Process -Name win*, Pow*

Sort-Object:-
The Cmdlet sorts objects in ascending & descending order based on the Parameter mentioned.

↳ Get-Process | Sort-Object -Property CPU -Descending →
for showing Roles in Descending order.

Measure-Object:-
* The Measure-Object Cmdlet Performs Calculations on the Property value of objects.
* It is used to Calculate the min, max, Sum & Avg of numeric values & also Count number of objects.

↳ Get-Process | Measure-Object

Each Property can mesured to calculate min, Max, Sum & Avg values.

↳ Get-Process | Measure-Object -Property VM -Sum -Average -maximum -minimum.

↳ Get-Process | Measure-Object -Property VM -Line -Word -Character

→ for showing no.of lines, word & charter of Process.

→ Fetch total no.of Processes running in system & highest CPU being Consumed.

↳ Get-Process | Measure-Object -Property CPU -Maximum | Select Count, Maximum.

Get-Content :-

The Get-Content Cmdlet retrives the Contents of a file from Specified location.

↳ Get-Content -Path myfile.txt → for showing content of file.

↳ Get-Process | Export-CSV myfile.csv → Get Process showing in .csv file format.

↳ Import-CSV -Path myfile.csv → showing tabular format data after import.

Get-Eventlog :-

It is used to retrive the events in event log.

↳ Get-Eventlog -list → shows Eventlogs all.

↳ Get-Eventlog -logname Application → for retriving the logs for Particular 'Application' Event.

→ Shows those events having error in 'application' event log.

↳ Get-Eventlog -logname Application -EntryType Error

formatting Data:- window PowerShell has a set of cmdlets that are used to format the way in which the Properties of object displayed.

The format cmdlets : format-wide, Format-list, Format-Table

↳ Format-wide only display single Property as it's -Property Parameter only takes a single value.

→Get-Eventlog -logname Application | Format-wide -Property Source.

↳ -Column Parameter can be used to specify no of column in which data has to be displayed.

→Get-Eventlog -logname Application | Format-wide -Property Source -Column 2

↳ Displays an object form of listing,

→Get-Eventlog -logname Application | Format-list

↳ the format-Table cmdlet formats the output in tabular format.

→Get-Eventlog -logname Application | Select Source, InstanceId -first 10

Source	InstanceId
:	:

→ tabular format showing

↳ retrieve the source details of application Eventlog having errors.

→Import-Csv -Path myfile.csv | where-Object {$_.EntryType -eq "Error"} | Select-Object Source

↳ formatting the data to be displayed in 5 column & redirecting to a text file.

→ Import-Csv -Path myfile.csv | Where-Object {$_.EntryType -eq "Error"} | Select-Object Source | Format-Wide -Column 5 | Out-File -FilePath MyFile.txt

Get-ChildItem :-

It is used to retrieve all the files & folder from any specified directory.

↳ Get-ChildItem → shows Path of current working dir.

↳ Get-ChildItem -Path C: -Recurse → for 'C' drive all files & folder showing.

→ for Creating new directory,

↳ New-Item -Name folder -Type Directory

→ for Creating new file,

↳ New-Item -Name file -Type File

→ To add Some value for Particular file while Creating,

↳ New-Item -Name file1 -Type File -value "welcome PS".

→ Copy the files or folder from one location to another,

↳ Copy-Item C:\MyData\file1 Destination C:\MyData\Fold\file1

→ Move-Item Cmdlet moves an item, including its Properties, Contents from one location to another.

↳ Move-Item file1 -Destination C:\MyData\Fold\file1.

→ Remove-Item Cmdlet is used for deletes the Specific Item.

↳ Remove-Item -Path "C:\MyData\file1"