Powershell - Create Folder

[**Previous Page**](https://www.tutorialspoint.com/powershell/powershell_files_folders.htm)

[**Next Page**](https://www.tutorialspoint.com/powershell/powershell_files_folders.htm)

Cmdlet

**New-Item** cmdlet is used to create a directory by passing the path using -Path as path of the directory and -ItemType as Directory.

Example

In this example, we'll create a folder in D:\Temp\ with name "Test Folder"

Type the following command in PowerShell ISE Console

New-Item -Path 'D:\temp\Test Folder' -ItemType Directory

Output

You will see the following output.

Directory: D:\temp

Mode LastWriteTime Length Name

---- ------------- ------ ----

d---- 4/3/2018 7:06 PM Test Folder

Powershell - Create File

[**Previous Page**](https://www.tutorialspoint.com/powershell/powershell_files_folders.htm)

[**Next Page**](https://www.tutorialspoint.com/powershell/powershell_files_folders.htm)

Cmdlet

**New-Item** cmdlet is used to create a file by passing the path using -Path as path of the file and -ItemType as File.

Example

In this example, we'll create a file in D:\Temp\Test Folder with name "Test File.txt"

Type the following command in PowerShell ISE Console

New-Item -Path 'D:\temp\Test Folder\Test File.txt' -ItemType File

Output

You will see the following output.

Directory: D:\temp

Mode LastWriteTime Length Name

---- ------------- ------ ----

-a--- 4/3/2018 7:14 PM 0 Test File.txt

Powershell - Copy Folder

[**Previous Page**](https://www.tutorialspoint.com/powershell/powershell_files_folders.htm)

[**Next Page**](https://www.tutorialspoint.com/powershell/powershell_files_folders.htm)

Cmdlet

**Copy-Item** cmdlet is used to copy a directory by passing the path of the directory to be copied and destination path where the folder is to be copied.

Example 1

In this example, we'll copy a folder D:\Temp\Test Folder as D:\Temp\Test Folder1

Type the following command in PowerShell ISE Console

Copy-Item 'D:\temp\Test Folder' 'D:\temp\Test Folder1'

You can see the Test Folder1 in Windows Explorer created.

Example 2

In this example, we'll copy a folder recursively D:\Temp\Test Folder to D:\Temp\Test Folder1

Type the following command in PowerShell ISE Console

Copy-Item 'D:\temp\Test Folder' -Destination 'D:\temp\Test Folder1'

Powershell - Copy File

[**Previous Page**](https://www.tutorialspoint.com/powershell/powershell_files_folders.htm)

[**Next Page**](https://www.tutorialspoint.com/powershell/powershell_files_folders.htm)

Cmdlet

**Copy-Item** cmdlet is used to copy a file by passing the path of the file to be copied and destination path where the file is to be copied.

Example 1

In this example, we'll copy a folder D:\Temp\Test Folder\Test File.txt to D:\Temp\Test Folder1

Type the following command in PowerShell ISE Console

Copy-Item 'D:\temp\Test Folder\Test File.txt' 'D:\temp\Test Folder1\Test File1.txt'

You can see the Test File1.txt in Test Folder1 with content of Test File.txt. Test Folder1 folder should be present before running this command.

Example 2

In this example, we'll copy all text file recursively D:\Temp\Test Folder to D:\Temp\Test Folder1

Type the following command in PowerShell ISE Console

Copy-Item -Filter \*.txt -Path 'D:\temp\Test Folder' -Recurse -Destination 'D:\temp\Test Folder1'

Powershell - Delete Folder

[**Previous Page**](https://www.tutorialspoint.com/powershell/powershell_files_folders.htm)

[**Next Page**](https://www.tutorialspoint.com/powershell/powershell_files_folders.htm)

Cmdlet

**Remove-Item** cmdlet is used to delete a directory by passing the path of the directory to be deleted.

Example 1

In this example, we'll delete a folder D:\Temp\Test Folder1

Type the following command in PowerShell ISE Console

Remove-Item 'D:\temp\Test Folder1'

You can see the Test Folder1 in Windows Explorer is deleted now.

Example 2

In this example, we'll remove the folder D:\Temp\Test Folder1 recursively. In first example, PowerShell confirms if directory is not empty. In this case, it will simply delete the item.

Type the following command in PowerShell ISE Console

Remove-Item 'D:\temp\Test Folder' -Recurse

Powershell - Delete File

[**Previous Page**](https://www.tutorialspoint.com/powershell/powershell_files_folders.htm)

[**Next Page**](https://www.tutorialspoint.com/powershell/powershell_files_folders.htm)

Cmdlet

**Remove-Item** cmdlet is used to delete a file by passing the path of the file to be deleted.

Example 1

In this example, we'll delete a file D:\Temp\Test Folder\Test.txt

Type the following command in PowerShell ISE Console

Remove-Item 'D:\temp\Test Folder\test.txt'

You can see the Test Folder1 in Windows Explorer is deleted now.

Example 2

In this example, we'll remove the folder D:\Temp\Test Folder recursively deleting its all files. In first example, PowerShell confirms if directory is not empty. In this case, it will simply delete the item.

Type the following command in PowerShell ISE Console

Remove-Item 'D:\temp\Test Folder' -Recurse

Powershell - Move Folder

[**Previous Page**](https://www.tutorialspoint.com/powershell/powershell_files_folders.htm)

[**Next Page**](https://www.tutorialspoint.com/powershell/powershell_files_folders.htm)

Cmdlet

**Move-Item** cmdlet is used to move a directory by passing the path of the directory to be moved and destination path where the folder is to be moved.

Example 1

In this example, we'll move a folder D:\Temp\Test to D:\Temp\Test1

Type the following command in PowerShell ISE Console

Move-Item D:\temp\Test D:\temp\Test1

You can see the Test directory moved to Test1 directory in Windows Explorer.

Example 2

In this example, Create a file test.txt in Test folder in D:\Temp\ and then run the same command.

Type the following command in PowerShell ISE Console

Move-Item D:\temp\Test D:\temp\Test1

Powershell - Move File

[**Previous Page**](https://www.tutorialspoint.com/powershell/powershell_files_folders.htm)

[**Next Page**](https://www.tutorialspoint.com/powershell/powershell_files_folders.htm)

Cmdlet

**Move-Item** cmdlet is used to move a file by passing the path of the file to be moved and destination path where the file is to be moved.

Example 1

In this example, we'll move a folder D:\Temp\Test\Test.txt to D:\Temp\Test1

Type the following command in PowerShell ISE Console

Move-Item D:\temp\Test\Test.txt D:\temp\Test1

Powershell - Rename Folder

[**Previous Page**](https://www.tutorialspoint.com/powershell/powershell_files_folders.htm)

[**Next Page**](https://www.tutorialspoint.com/powershell/powershell_files_folders.htm)

Cmdlet

**Rename-Item** cmdlet is used to rename a folder by passing the path of the folder to be renamed and target name.

Example 1

In this example, we'll rename a folder D:\Temp\Test to D:\Temp\Test1

Type the following command in PowerShell ISE Console

Rename-Item "D:\temp\Test Test1"

Powershell - Rename File

[**Previous Page**](https://www.tutorialspoint.com/powershell/powershell_files_folders.htm)

[**Next Page**](https://www.tutorialspoint.com/powershell/powershell_files_folders.htm)

Cmdlet

**Rename-Item** cmdlet is used to rename a File by passing the path of the file to be renamed and target name.

Example 1

In this example, we'll rename a folder D:\Temp\Test\test.txt to test1.txt

Type the following command in PowerShell ISE Console

Rename-Item D:\temp\Test\test.txt test1.txt

Powershell - Retrieving Item

[**Previous Page**](https://www.tutorialspoint.com/powershell/powershell_files_folders.htm)

[**Next Page**](https://www.tutorialspoint.com/powershell/powershell_files_folders.htm)

Cmdlet

**Get-Content** cmdlet is used to retrieve content of a file as an array.

Example 1

In this example, we'll read a file D:\Temp\Test\Test.txt

Type the following command in PowerShell ISE Console

Get-Content D:\temp\Test\test.txt

Output

You can see following output in PowerShell console.

Get-Content D:\temp\test\test.txt

;This is a test file.

Example 2

In this example, we'll read the size of the content of the file read.

Type the following command in PowerShell ISE Console

(Get-Content D:\temp\test\test.txt).length

You can see following output in PowerShell console.

(Get-Content D:\temp\test\test.txt).length

20

Powershell - Check Folder Existence

[**Previous Page**](https://www.tutorialspoint.com/powershell/powershell_files_folders.htm)

[**Next Page**](https://www.tutorialspoint.com/powershell/powershell_files_folders.htm)

Cmdlet

**Test-Path** cmdlet is used to check existence of a folder.

Example 1

In this example, we're having a folder test in D:\temp directory

Type the following command in PowerShell ISE Console

Test-Path D:\temp\test

Output

You can see following output in PowerShell console.

Test-Path D:\temp\test

True

Example 2

In this example, we're not having a folder named test2 in D:\temp directory

Type the following command in PowerShell ISE Console

Test-Path D:\temp\test2

Output

You can see following output in PowerShell console.

Test-Path D:\temp\test2

False

Powershell - Check File Existence

[**Previous Page**](https://www.tutorialspoint.com/powershell/powershell_files_folders.htm)

[**Next Page**](https://www.tutorialspoint.com/powershell/powershell_files_folders.htm)

Cmdlet

**Test-Path** cmdlet is used to check existence of a file.

Example 1

In this example, we're having a file test.txt in D:\temp\test directory

Type the following command in PowerShell ISE Console

Test-Path D:\temp\test\test.txt

Output

You can see following output in PowerShell console.

Test-Path D:\temp\test\test.txt

True

Example 2

In this example, we're not having a file named test2.txt in D:\temp\test directory

Type the following command in PowerShell ISE Console

Test-Path D:\temp\test\test2.txt

Output

You can see following output in PowerShell console.

Test-Path D:\temp\test\test2.txt

False

Powershell - Set System Date

[**Previous Page**](https://www.tutorialspoint.com/powershell/powershell_dates_timers.htm)

[**Next Page**](https://www.tutorialspoint.com/powershell/powershell_dates_timers.htm)

Cmdlet

**Set-Date** cmdlet is used to set System Date.

In this example, we're using Get-Date to get current date

Type the following command in PowerShell ISE Console

Get-Date

Output

You can see following output in PowerShell console.

Get-Date

Saturday, May 05, 2018 9:58:06 AM

In this example, we're using Set-Date to add one more day to current date.

Type the following command in PowerShell ISE Console

set-date -Date (Get-Date).AddDays(1)

Output

You can see following output in PowerShell console.

Sunday, May 06, 2018 9:59:16 AM

Now revert back to substract added day to current date.

Type the following command in PowerShell ISE Console

set-date -Date (Get-Date).AddDays(-1)

Output

You can see following output in PowerShell console.

Saturday, May 05, 2018 10:00:37 AM

Powershell - Get System Date

[**Previous Page**](https://www.tutorialspoint.com/powershell/powershell_dates_timers.htm)

[**Next Page**](https://www.tutorialspoint.com/powershell/powershell_dates_timers.htm)

Cmdlet

**Get-Date** cmdlet is used to get System Date.

Example 1

In this example, we're using Get-Date without any parameter

Type the following command in PowerShell ISE Console

Get-Date

Output

You can see following output in PowerShell console.

Get-Date

Wednesday, April 04, 2018 5:24:51 PM

Example 2

In this example, we're using -DisplayHint to print only Date.

Type the following command in PowerShell ISE Console

Get-Date -DisplayHint Date

Output

You can see following output in PowerShell console.

Get-Date -DisplayHint Date

Wednesday, April 04, 2018

Powershell - Get System Time

[**Previous Page**](https://www.tutorialspoint.com/powershell/powershell_dates_timers.htm)

[**Next Page**](https://www.tutorialspoint.com/powershell/powershell_dates_timers.htm)

Cmdlet

**Get-Date** cmdlet is used to get System Date-Time.

Example 1

In this example, we're using Get-Date without any parameter

Type the following command in PowerShell ISE Console

Get-Date

Output

You can see following output in PowerShell console.

Get-Date

Wednesday, April 04, 2018 5:24:51 PM

Example 2

In this example, we're using -DisplayHint to print only Time.

Type the following command in PowerShell ISE Console

Get-Date -DisplayHint Time

Output

You can see following output in PowerShell console.

Get-Date -DisplayHint Date

5:26:36 PM

Powershell - Set System Time

[**Previous Page**](https://www.tutorialspoint.com/powershell/powershell_dates_timers.htm)

[**Next Page**](https://www.tutorialspoint.com/powershell/powershell_dates_timers.htm)

Cmdlet

**Set-Date** cmdlet is used to set System Time.

In this example, we're using Get-Date to get current date

Type the following command in PowerShell ISE Console

get-date -displayHint time

Output

You can see following output in PowerShell console.

10:04:18 AM

In this example, we're using Set-Date to adjust to 60 minutes.

Type the following commands in PowerShell ISE Console

> $timeToAdd = New-TimeSpan -Minutes -60

> set-date -adjust $timeToAdd

Output

You can see following output in PowerShell console.

Saturday, May 05, 2018 11:05:16 AM

Now revert back to substract added time to current date.

Type the following command in PowerShell ISE Console

> $timeToAdd = New-TimeSpan -Minutes -60

> set-date -adjust $timeToAdd

Output

You can see following output in PowerShell console.

Saturday, May 05, 2018 10:08:54 AM

Powershell - Create Text File

[**Previous Page**](https://www.tutorialspoint.com/powershell/powershell_files_io.htm)

[**Next Page**](https://www.tutorialspoint.com/powershell/powershell_files_io.htm)

Cmdlet

**New-Item** cmdlet is used to create a text file and **Set-Content** cmdlet to put content into it.

Step 1

In this example, we're creating a new text file named test.txt

Type the following command in PowerShell ISE Console

New-Item D:\temp\test\test.txt

You can see the test.txt created in D:\temp\test directory.

Step 2

In this example, we're adding content to test.txt.

Type the following command in PowerShell ISE Console

Set-Content D:\temp\test\test.txt 'Welcome to '

Step 3

In this example, we're reading content of test.txt.

get-Content D:\temp\test\test.txt

Output

You can see following output in PowerShell console.

Welcome to

Powershell - Read Text File

[**Previous Page**](https://www.tutorialspoint.com/powershell/powershell_files_io.htm)

[**Next Page**](https://www.tutorialspoint.com/powershell/powershell_files_io.htm)

Cmdlet

**Get-Content** cmdlet is used to read content of a text file.

In this example, we're reading content of test.txt.

Get-Content D:\temp\test\test.txt

Output

You can see following output in PowerShell console.

Welcome to

Powershell - Create XML File

[**Previous Page**](https://www.tutorialspoint.com/powershell/powershell_files_io.htm)

[**Next Page**](https://www.tutorialspoint.com/powershell/powershell_files_io.htm)

Cmdlet

**New-Item** cmdlet is used to create a xml file and **Set-Content** cmdlet to put content into it.

Step 1

In this example, we're creating a new xml file named test.xml

Type the following command in PowerShell ISE Console

New-Item D:\temp\test\test.xml -ItemType File

You can see the test.xml created in D:\temp\test directory.

Step 2

In this example, we're adding content to test.xml.

Type the following command in PowerShell ISE Console

Set-Content D:\temp\test\test.xml '<title>Welcome to </title>'

Step 3

In this example, we're reading content of test.xml.

Get-Content D:\temp\test\test.xml

Output

You can see following output in PowerShell console.

<title>Welcome to </title>

Powershell - Read XML File

[**Previous Page**](https://www.tutorialspoint.com/powershell/powershell_files_io.htm)

[**Next Page**](https://www.tutorialspoint.com/powershell/powershell_files_io.htm)

Cmdlet

**Get-Content** cmdlet is used to read content of a xml file.

In this example, we're reading content of test.xml.

Get-Content D:\temp\test\test.xml

Output

You can see following output in PowerShell console.

<title>Welcome to </title>

Powershell - Create CSV File

[**Previous Page**](https://www.tutorialspoint.com/powershell/powershell_files_io.htm)

[**Next Page**](https://www.tutorialspoint.com/powershell/powershell_files_io.htm)

Cmdlet

**New-Item** cmdlet is used to create a csv file and **Set-Content** cmdlet to put content into it.

Step 1

In this example, we're creating a new csv file named test.csv

Type the following command in PowerShell ISE Console

New-Item D:\temp\test\test.csv -ItemType File

You can see the test.csv created in D:\temp\test directory.

Step 2

In this example, we're adding content to test.csv.

Type the following command in PowerShell ISE Console

Set-Content D:\temp\test\test.csv 'Mahesh,Suresh,Ramesh'

Step 3

In this example, we're reading content of test.csv.

Get-Content D:\temp\test\test.csv

Output

You can see following output in PowerShell console.

Mahesh,Suresh,Ramesh

Powershell - Read CSV File

[**Previous Page**](https://www.tutorialspoint.com/powershell/powershell_files_io.htm)

[**Next Page**](https://www.tutorialspoint.com/powershell/powershell_files_io.htm)

Cmdlet

**Get-Content** cmdlet is used to read content of a csv file.

In this example, we're reading content of test.csv.

Get-Content D:\temp\test\test.csv

Output

You can see following output in PowerShell console.

Mahesh,Suresh,Ramesh

Powershell - Create HTML File

[**Previous Page**](https://www.tutorialspoint.com/powershell/powershell_files_io.htm)

[**Next Page**](https://www.tutorialspoint.com/powershell/powershell_files_io.htm)

Cmdlet

**New-Item** cmdlet is used to create a html file and **Set-Content** cmdlet to put content into it.

Step 1

In this example, we're creating a new html file named test.html

Type the following command in PowerShell ISE Console

New-Item D:\temp\test\test.html -ItemType File

You can see the test.html created in D:\temp\test directory.

Step 2

In this example, we're adding content to test.html.

Type the following command in PowerShell ISE Console

Set-Content D:\temp\test\test.html '<html>Welcome to </html>'

Step 3

In this example, we're reading content of test.html.

Get-Content D:\temp\test\test.html

Output

You can see following output in PowerShell console.

<html>Welcome to </html>

Powershell - Read HTML File

[**Previous Page**](https://www.tutorialspoint.com/powershell/powershell_files_io.htm)

[**Next Page**](https://www.tutorialspoint.com/powershell/powershell_files_io.htm)

Cmdlet

**Get-Content** cmdlet is used to read content of a html file.

In this example, we're reading content of test.html.

Get-Content D:\temp\test\test.html

Output

You can see following output in PowerShell console.

<html>Welcome to </html>

Powershell - Erase content of File

[**Previous Page**](https://www.tutorialspoint.com/powershell/powershell_files_io.htm)

[**Next Page**](https://www.tutorialspoint.com/powershell/powershell_files_io.htm)

Cmdlet

**Clear-Content** cmdlet can be used to erase content of a txt file.

In this example, we're erasing content of test.txt.

Clear-Content D:\temp\test\test.txt

Now, if we read content of test.txt.

Get-Content D:\temp\test\test.txt

Powershell - Append content to File

[**Previous Page**](https://www.tutorialspoint.com/powershell/powershell_files_io.htm)

[**Next Page**](https://www.tutorialspoint.com/powershell/powershell_files_io.htm)

Cmdlet

**Add-Content** cmdlet can be used to append content of a any file.

In this example, we're adding content of test.txt.

Step 1

In this example, we're set content in a new txt file named test.txt

Type the following command in PowerShell ISE Console

Set-Content D:\temp\test\test.txt 'Hello'

Step 2

In this example, we're appending content to test.html.

Type the following command in PowerShell ISE Console

Add-Content D:\temp\test\test.txt 'World!'

Step 3

In this example, we're reading content of test.html.

Get-Content D:\temp\test\test.txt

Output

You can see following output in PowerShell console.

Hello

World!

## Cmdlets

A cmdlet or "Command let" is a lightweight command used in the Windows PowerShell environment. The Windows PowerShell runtime invokes these cmdlets at command prompt. You can create and invoke them programmatically through Windows PowerShell APIs. Following are advanced usage example of cmdlets.

## Cmdlet

**Get-Unique** cmdlet can be used to get the unique objects from a sorted list of objects.

In this example, we're see the Get-Unique cmdlet in action.

## Step 1

In this example, we're set list of strings in a variable.

Type the following command in PowerShell ISE Console

$list = "one","two","two","three","four","five"

## Step 2

In this example, we're printing the original list of strings.

Type the following command in PowerShell ISE Console

$list

## Output

You can see following output in PowerShell console.

one

two

two

three

four

five

## Step 3

In this example, we're sorting the list and then get the unique values.

Type the following command in PowerShell ISE Console

$list | sort | get-unique

## Output

You can see following output in PowerShell console.

five

four

one

three

two

Powershell - Measure-Object Cmdlet

[**Previous Page**](https://www.tutorialspoint.com/powershell/powershell_advanced_cmdlets.htm)

[**Next Page**](https://www.tutorialspoint.com/powershell/powershell_advanced_cmdlets.htm)

Cmdlet

**Measure-Object** cmdlet can be used to get the properties of the passed output such as min, max, size, count, line etc.

In these examples, we're see the Measure-Object cmdlet in action.

Example 1

In this example, first we've a file test.txt in D:\temp\test with content "Welcome to .Com".

Type the following command in PowerShell ISE Console

get-content D:\temp\test\test.txt | measure-object -character -line -word

Output

You can see following output in PowerShell console.

Lines Words Characters Property

----- ----- ---------- --------

1 3 29

Example 2

In this example, We'll count the no. of files present in current directory.

Type the following command in PowerShell ISE Console

Get-ChildItem | Measure-Object

Output

You can see following output in PowerShell console.

Count : 25

Average :

Sum :

Maximum :

Minimum :

Property :

Powershell - Compare-Object Cmdlet

[**Previous Page**](https://www.tutorialspoint.com/powershell/powershell_advanced_cmdlets.htm)

[**Next Page**](https://www.tutorialspoint.com/powershell/powershell_advanced_cmdlets.htm)

**Compare-Object** cmdlet can be used to compare two objects.

In these examples, we're see the Compare-Object cmdlet in action.

Example 1

In this example, first we've a file test.txt in D:\temp\test with content "Welcome to .Com" and test1.txt with content "Hello World!" and "Welcome to .Com" in two lines.

Compare the files. Type the following command in PowerShell ISE Console. Common line(s) will be displayed.

Compare-Object -ReferenceObject $(Get-Content D:\temp\test\test.txt) -DifferenceObject $(Get-Content D:\temp\test\test1.txt)

Output

You can see following output in PowerShell console.

InputObject SideIndicator

----------- -------------

Hello World! =>

Example 2

Compare the content of files. Type the following command in PowerShell ISE Console. All line(s) with indicator will be displayed.

Type the following command in PowerShell ISE Console

Compare-Object -ReferenceObject $(Get-Content D:\temp\test\test.txt) -DifferenceObject $(Get-Content D:\temp\test\test1.txt) -IncludeEqual

Output

You can see following output in PowerShell console.

InputObject SideIndicator

----------- -------------

Welcome to .Com ==

Hello World! =>

Powershell - Format-List Cmdlet

[**Previous Page**](https://www.tutorialspoint.com/powershell/powershell_advanced_cmdlets.htm)

[**Next Page**](https://www.tutorialspoint.com/powershell/powershell_advanced_cmdlets.htm)

cmdlet

**Format-List** cmdlet can be used to formats the output as a list of properties where a property appears on a new line.

In these examples, we're see the Format-List cmdlet in action.

Example 1

In this example, first we've a file test.txt in D:\temp\test with content "Welcome to .Com" and test1.txt with content "Hello World!" and "Welcome to .Com" in two lines.

Get the file details in a variable.

$A = Get-ChildItem D:\temp\test\\*.txt

Get the file details using Format-List cmdlet.

Format-List -InputObject $A

Output

You can see following output in PowerShell console.

Directory: D:\temp\test

Name : test.txt

Length : 31

CreationTime : 4/4/2018 4:48:38 PM

LastWriteTime : 4/11/2018 4:40:15 PM

LastAccessTime : 4/4/2018 4:48:38 PM

VersionInfo : File: D:\temp\test\test.txt

InternalName:

OriginalFilename:

FileVersion:

FileDescription:

Product:

ProductVersion:

Debug: False

Patched: False

PreRelease: False

PrivateBuild: False

SpecialBuild: False

Language:

Name : test1.txt

Length : 44

CreationTime : 4/12/2018 6:54:48 PM

LastWriteTime : 4/12/2018 6:56:21 PM

LastAccessTime : 4/12/2018 6:54:48 PM

VersionInfo : File: D:\temp\test\test1.txt

InternalName:

OriginalFilename:

FileVersion:

FileDescription:

Product:

ProductVersion:

Debug: False

Patched: False

PreRelease: False

PrivateBuild: False

SpecialBuild: False

Language:

Example 2

Get the list of services

Type the following command in PowerShell ISE Console

Get-Service | Format-List

Output

You can see following output in PowerShell console.

Name : AdobeARMservice

DisplayName : Adobe Acrobat Update Service

Status : Running

DependentServices : {}

ServicesDependedOn : {}

CanPauseAndContinue : False

CanShutdown : False

CanStop : True

ServiceType : Win32OwnProcess

Name : AdobeFlashPlayerUpdateSvc

DisplayName : Adobe Flash Player Update Service

Status : Stopped

DependentServices : {}

ServicesDependedOn : {}

CanPauseAndContinue : False

CanShutdown : False

CanStop : False

ServiceType : Win32OwnProcess

Powershell - Format-Wide Cmdlet

[**Previous Page**](https://www.tutorialspoint.com/powershell/powershell_advanced_cmdlets.htm)

[**Next Page**](https://www.tutorialspoint.com/powershell/powershell_advanced_cmdlets.htm)

cmdlet

**Format-Wide** cmdlet can be used to formats the output as a table with one property per object.

In these examples, we're see the Format-Wide cmdlet in action.

Example 1

In this example, first we've a file test.txt in D:\temp\test with content "Welcome to .Com" and test1.txt with content "Hello World!" and "Welcome to .Com" in two lines.

Get the file details in a variable.

$A = Get-ChildItem D:\temp\test\\*.txt

Get the file details using Format-Wide cmdlet.

Format-Wide -InputObject $A

Output

You can see following output in PowerShell console.

Directory: D:\temp\test

test.txt test1.txt

Example 2

Get the required property.

Type the following command in PowerShell ISE Console

Format-Wide -InputObject $A -Property -Property Length

Output

You can see following output in PowerShell console.

Directory: D:\temp\test

31 44

Powershell - Where-Object Cmdlet

[**Previous Page**](https://www.tutorialspoint.com/powershell/powershell_advanced_cmdlets.htm)

[**Next Page**](https://www.tutorialspoint.com/powershell/powershell_advanced_cmdlets.htm)

cmdlet

**Where-Object** cmdlet can be used to select objects having particular property values from the collection of objects that are passed to it.

In these examples, we're see the Where-Object cmdlet in action.

Example 1

Get stopped services.

Get-Service | Where-Object {$\_.Status -eq "Stopped"}

Output

You can see following output in PowerShell console.

Status Name DisplayName

------ ---- -----------

Stopped AdobeFlashPlaye... Adobe Flash Player Update Service

Stopped AeLookupSvc Application Experience

Example 2

Get processes based on process name.

Type the following command in PowerShell ISE Console

Get-Process | Where-Object {$\_.ProcessName -Match "^p.\*"}

Powershell - Get-ChildItem Cmdlet

[**Previous Page**](https://www.tutorialspoint.com/powershell/powershell_advanced_cmdlets.htm)

[**Next Page**](https://www.tutorialspoint.com/powershell/powershell_advanced_cmdlets.htm)

Cmdlet

**Get-ChildItem** cmdlet can be used to get the items or child items in one or more specific locations.

In these examples, we're see the Get-ChildItem cmdlet in action.

Example 1

In this example, first we've a file test.txt in D:\temp\test with content "Welcome to .Com" and test1.txt with content "Hello World!" and "Welcome to .Com" in two lines.

Get the file details in a variable.

$A = Get-ChildItem D:\temp\test\\*.txt

Get the file details using Format-Wide cmdlet.

Format-Wide -InputObject $A

Output

You can see following output in PowerShell console.

Directory: D:\temp\test

test.txt test1.txt

Example 2

Get the names of the items in current directory.

Type the following command in PowerShell ISE Console

Get-ChildItem -Name

Output

You can see following output in PowerShell console consider being in D:\temp\Test directory.

test.csv

test.txt

test.xml

test1.txt

Powershell - ForEach-Object Cmdlet

[**Previous Page**](https://www.tutorialspoint.com/powershell/powershell_advanced_cmdlets.htm)

[**Next Page**](https://www.tutorialspoint.com/powershell/powershell_advanced_cmdlets.htm)

Cmdlet

**ForEach-Object** cmdlet can be used to perform operations on each object of a collection of objects.

In these examples, we're see the ForEach-Object cmdlet in action.

Example 1

In this example, we'll divide integer in an array. We'll refer to each object using $\_.

1000,2000,3000 | ForEach-Object -Process {$\_/1000}

Output

You can see following output in PowerShell console.

1

2

3

Example 2

Get the names of the items in current directory.

In this example, we'll split powershell module names.

"Microsoft.PowerShell.Core", "Microsoft.PowerShell.Host" | ForEach-Object {$\_.Split(".")}

Output

You can see following output in PowerShell console.

Microsoft

PowerShell

Core

Microsoft

PowerShell

Host

Powershell - Start-Sleep Cmdlet

[**Previous Page**](https://www.tutorialspoint.com/powershell/powershell_advanced_cmdlets.htm)

[**Next Page**](https://www.tutorialspoint.com/powershell/powershell_advanced_cmdlets.htm)

Cmdlet

**Start-Sleep** cmdlet suspends the activity in a script or session for the particular period of time.

In these examples, we're see the Start-Sleep cmdlet in action.

Example 1

In this example, we'll suspend the current process for 15 seconds.

Start-Sleep -s 15

Output

You can see PowerShell console resumes after 15 seconds.

Example 2

In this example, we'll suspend the current process for 500 milliseconds.

Start-Sleep -m 500

Powershell - Read-Host Cmdlet

[**Previous Page**](https://www.tutorialspoint.com/powershell/powershell_advanced_cmdlets.htm)

[**Next Page**](https://www.tutorialspoint.com/powershell/powershell_advanced_cmdlets.htm)

Cmdlet

**Read-Host** cmdlet is used to read from the console.

In these example, we're see the Read-Host cmdlet in action.

Example

In this example, we'll ask the user to pass an input and read the input into a variable.

$choice = Read-Host "Please put your choice"

Powershell will show a popup to enter the value. Once you enter the value, it is saved in $choice variable. Now print the variable.

$choice

Output

You can see the value of variable.

1

Powershell - Select-Object Cmdlet

[**Previous Page**](https://www.tutorialspoint.com/powershell/powershell_advanced_cmdlets.htm)

[**Next Page**](https://www.tutorialspoint.com/powershell/powershell_advanced_cmdlets.htm)

Cmdlet

**Select-Object** cmdlet is used to select object or its properties.

In these example, we're see the Select-Object cmdlet in action.

Example 1

In this example, we'll create objects using Process properties.

Get-Process | Select-Object -Property ProcessName, Id, WS -Last 5

Output

You can see the following output.

ProcessName Id WS

----------- -- --

UNS 2624 10301440

wininit 624 4935680

winlogon 552 7774208

WLTRYSVC 3080 3608576

WmiPrvSE 1620 11870208

Example 2

In this example, we'll select unique values of an array.

"a","b","c","a","a","a" | Select-Object -Unique

Output

You can see the following output.

a

b

c

Powershell - Sort-Object Cmdlet

[**Previous Page**](https://www.tutorialspoint.com/powershell/powershell_advanced_cmdlets.htm)

[**Next Page**](https://www.tutorialspoint.com/powershell/powershell_advanced_cmdlets.htm)

Cmdlet

**Sort-Object** cmdlet is used to sort object by its properties.

In these example, we're see the Sort-Object cmdlet in action.

Example 1

In this example, we'll sort objects using Process properties.

Get-Process | Sort-Object -Property WS | Select-Object -Last 5

Output

You can see the process with high memory usages.

Handles NPM(K) PM(K) WS(K) VM(M) CPU(s) Id ProcessName

------- ------ ----- ----- ----- ------ -- -----------

314 44 134528 108492 760 28.38 1536 powershell\_ise

579 25 116552 124832 205 21.68 256 svchost

1249 59 77132 130152 433 41.90 4392 chrome

329 42 104748 133704 1935 59.22 4368 chrome

604 67 163376 149552 277 0.45 3152 mysqld

Example 2

In this example, we'll sort an array.

"d","e","c","a","b","f" | Sort-Object

Output

You can see the following output.

a

b

c

d

e

f

Powershell - Write-Warning Cmdlet

[**Previous Page**](https://www.tutorialspoint.com/powershell/powershell_advanced_cmdlets.htm)

[**Next Page**](https://www.tutorialspoint.com/powershell/powershell_advanced_cmdlets.htm)

Cmdlet

**Write-Warning** cmdlet is used to write warning messages.

In these example, we're see the Write-Warning cmdlet in action.

Example

In this example, we'll show a warning message.

Write-Warning "Test Warning"

Output

You can see the process with high memory usages.

WARNING: Test Warning

Powershell - Write-Host Cmdlet

[**Previous Page**](https://www.tutorialspoint.com/powershell/powershell_advanced_cmdlets.htm)

[**Next Page**](https://www.tutorialspoint.com/powershell/powershell_advanced_cmdlets.htm)

Cmdlet

**Write-Host** cmdlet is used to write customized messages.

In these example, we're see the Write-Host cmdlet in action.

Example

In this example, we'll show a customized message.

Write-Host (2,4,6,8,10,12) -Separator ", -> " -ForegroundColor DarkGreen -BackgroundColor White

Output

You can see the even numbers with separator, in green color and on white background.

2, -> 4, -> 6, -> 8, -> 10, -> 12

Powershell - Invoke-Item Cmdlet

[**Previous Page**](https://www.tutorialspoint.com/powershell/powershell_advanced_cmdlets.htm)

[**Next Page**](https://www.tutorialspoint.com/powershell/powershell_advanced_cmdlets.htm)

Cmdlet

**Invoke-Item** cmdlet is used to perform a default action on specified item.

In these example, we're see the Invoke-Item cmdlet in action.

Example

In this example, we'll show a customized message.

Invoke-Item "D:\Temp\test.txt"

Output

You can see text.txt open with notepad as it is invoked by Powershell as default action.

Powershell - Invoke-Expression Cmdlet

[**Previous Page**](https://www.tutorialspoint.com/powershell/powershell_advanced_cmdlets.htm)

[**Next Page**](https://www.tutorialspoint.com/powershell/powershell_advanced_cmdlets.htm)

Cmdlet

**Invoke-Expression** cmdlet is used to perform a command or expression on local computer.

In these example, we're see the Invoke-Expression cmdlet in action.

Example

In this example, we'll show how to invoke an expression.

> $Command = 'Get-Process'

> $Command

Get-Process

> Invoke-Expression $Command

Output

Here you can notice that $Command print the expression where as Invoke-Expression executes the expression.

Handles NPM(K) PM(K) WS(K) VM(M) CPU(s) Id ProcessName

------- ------ ----- ----- ----- ------ ---- -----------

78 8 1288 3892 43 0.02 2556 armsvc

119 9 15808 15572 50 600 audiodg

88 8 2512 5860 73 0.02 3144 chrome

186 19 23360 31580 740 0.23 3148 chrome

218 22 28432 44848 769 0.86 3492 chrome

owershell - Measure-Command Cmdlet

[**Previous Page**](https://www.tutorialspoint.com/powershell/powershell_advanced_cmdlets.htm)

[**Next Page**](https://www.tutorialspoint.com/powershell/powershell_advanced_cmdlets.htm)

Cmdlet

**Measure-Command** cmdlet is used to meaure the time taken by script or command.

In these example, we're see the Measure-Command cmdlet in action.

Example

In this example, we'll show how to measure time of Get-EventLog command to log an event in PowerShell event log.

Measure-Command { Get-EventLog "Windows PowerShell" }

Output

Days : 0

Hours : 0

Minutes : 0

Seconds : 0

Milliseconds : 50

Ticks : 506776

TotalDays : 5.86546296296296E-07

TotalHours : 1.40771111111111E-05

TotalMinutes : 0.000844626666666667

TotalSeconds : 0.0506776

TotalMilliseconds : 50.6776

Powershell - Invoke-History Cmdlet

[**Previous Page**](https://www.tutorialspoint.com/powershell/powershell_advanced_cmdlets.htm)

[**Next Page**](https://www.tutorialspoint.com/powershell/powershell_advanced_cmdlets.htm)

Cmdlet

**Invoke-History** cmdlet is used to run the command from the current session which are already run.

In these example, we're see the Invoke-History cmdlet in action.

Example

In this example, we'll show how to invoke last run command using Invoke-History. Call Invoke-History without parameter.

Invoke-History

Measure-Command { Get-EventLog "Windows PowerShell" }

Output

Days : 0

Hours : 0

Minutes : 0

Seconds : 0

Milliseconds : 11

Ticks : 116083

TotalDays : 1.34355324074074E-07

TotalHours : 3.22452777777778E-06

TotalMinutes : 0.000193471666666667

TotalSeconds : 0.0116083

TotalMilliseconds : 11.6083

Powershell - Add-History Cmdlet

[**Previous Page**](https://www.tutorialspoint.com/powershell/powershell_advanced_cmdlets.htm)

[**Next Page**](https://www.tutorialspoint.com/powershell/powershell_advanced_cmdlets.htm)

Cmdlet

**Add-History** cmdlet is used to add commands in current history.

In these example, we're see the Add-History cmdlet in action.

Example

In this example, we'll add first five history command to current history again.

> get-history

Id CommandLine

-- -----------

13 clear-history

14 get-history

15 dir

16 dir

17 dir

18 dir

> Get-history -count 5 | Add-history

Now get the history again to see the effect of Add-history.

> Get-history

Id CommandLine

-- -----------

13 clear-history

14 get-history

15 dir

16 dir

17 dir

18 dir

19 get-history

20 get-history -count 5 | Add-history

21 dir

22 dir

23 dir

24 get-history

25 get-history -count 5 | Add-history

Powershell - Get-History Cmdlet

[**Previous Page**](https://www.tutorialspoint.com/powershell/powershell_advanced_cmdlets.htm)

[**Next Page**](https://www.tutorialspoint.com/powershell/powershell_advanced_cmdlets.htm)

Cmdlet

**Get-History** cmdlet is used to get commands run in current session.

In these example, we're see the Get-History cmdlet in action.

Example

In this example, we'll get commands run in current history.

> get-history

Id CommandLine

-- -----------

13 clear-history

14 get-history

15 dir

16 dir

17 dir

18 dir

Powershell - Get-Culture Cmdlet

[**Previous Page**](https://www.tutorialspoint.com/powershell/powershell_advanced_cmdlets.htm)

[**Next Page**](https://www.tutorialspoint.com/powershell/powershell_advanced_cmdlets.htm)

Cmdlet

**Get-Culture** cmdlet is used to get current culture set in windows.

In these example, we're see the Get-Culture cmdlet in action.

Example

In this example, we'll get culture settings.

> get-culture

LCID Name DisplayName

---- ---- -----------

|  |  |
| --- | --- |
| **Operator** | **Description** |
| $$ | Represents the last token in the last line received by the session. |
| $? | Represents the execution status of the last operation. It contains TRUE if the last operation succeeded and FALSE if it failed. |
| $^ | Represents the first token in the last line received by the session. |
| $\_ | Same as $PSItem. Contains the current object in the pipeline object. You can use this variable in commands that perform an action on every object or on selected objects in a pipeline. |
| $ARGS | Represents an array of the undeclared parameters and/or parameter values that are passed to a function, script, or script block. |
| $CONSOLEFILENAME | Represents the path of the console file (.psc1) that was most recently used in the session. |
| $ERROR | Represents an array of error objects that represent the most recent errors. |
| $EVENT | Represents a PSEventArgs object that represents the event that is being processed. |
| $EVENTARGS | Represents an object that represents the first event argument that derives from EventArgs of the event that is being processed. |
| $EVENTSUBSCRIBER | Represents a PSEventSubscriber object that represents the event subscriber of the event that is being processed. |
| $EXECUTIONCONTEXT | Represents an EngineIntrinsics object that represents the execution context of the PowerShell host. |
| $FALSE | Represents FALSE. You can use this variable to represent FALSE in commands and scripts instead of using the string "false". |
| $FOREACH | Represents the enumerator (not the resulting values) of a ForEach loop. You can use the properties and methods of enumerators on the value of the $ForEach variable. |
| $HOME | Represents the full path of the user's home directory. |
| $HOST | Represents an object that represents the current host application for PowerShell. |
| $INPUT | Represents an enumerator that enumerates all input that is passed to a function. |
| $LASTEXITCODE | Represents the exit code of the last Windows-based program that was run. |
| $MATCHES | The $Matches variable works with the -match and -notmatch operators. |
| $MYINVOCATION | $MyInvocation is populated only for scripts, function, and script blocks. PSScriptRoot and PSCommandPath properties of the $MyInvocation automatic variable contain information about the invoker or calling script, not the current script. |
| $NESTEDPROMPTLEVEL | Represents the current prompt level. |
| $NULL | $null is an automatic variable that contains a NULL or empty value. You can use this variable to represent an absent or undefined value in commands and scripts. |
| $PID | Represents the process identifier (PID) of the process that is hosting the current PowerShell session. |
| $PROFILE | Represents the full path of the PowerShell profile for the current user and the current host application. |
| $PSCMDLET | Represents an object that represents the cmdlet or advanced function that is being run. |
| $PSCOMMANDPATH | Represents the full path and file name of the script that is being run. |
| $PSCULTURE | Represents the name of the culture currently in use in the operating system. |
| $PSDEBUGCONTEXT | While debugging, this variable contains information about the debugging environment. Otherwise, it contains a NULL value. |
| $PSHOME | Represents the full path of the installation directory for PowerShell. |
| $PSITEM | Same as $\_. Contains the current object in the pipeline object. |
| $PSSCRIPTROOT | Represents the directory from which a script is being run. |
| $PSSENDERINFO | Represents information about the user who started the PSSession, including the user identity and the time zone of the originating computer. |
| $PSUICULTURE | Represents the name of the user interface (UI) culture that is currently in use in the operating system. |
| $PSVERSIONTABLE | Represents a read-only hash table that displays details about the version of PowerShell that is running in the current session. |
| $SENDER | Represents the object that generated this event. |
| $SHELLID | Represents the identifier of the current shell. |
| $STACKTRACE | Represents a stack trace for the most recent error. |
| $THIS | In a script block that defines a script property or script method, the $This variable refers to the object that is being extended. |
| $TRUE | Represents TRUE. You can use this variable to represent TRUE in commands and scripts. |

Powershell - For Loop

[**Previous Page**](https://www.tutorialspoint.com/powershell/powershell_looping.htm)

[**Next Page**](https://www.tutorialspoint.com/powershell/powershell_looping.htm)

The following scripts demonstrates the for loop.

> $array = @("item1", "item2", "item3")

> for($i = 0; $i -lt $array.length; $i++){ $array[$i] }

item1

item2

item3

−

Powershell - ForEach Loop

[**Previous Page**](https://www.tutorialspoint.com/powershell/powershell_looping.htm)

[**Next Page**](https://www.tutorialspoint.com/powershell/powershell_looping.htm)

The following scripts demonstrates the ForEach loop.

> $array = @("item1", "item2", "item3")

> foreach ($element in $array) { $element }

item1

item2

item3

> $array | foreach { $\_ }

item1

item2

item3

Powershell - While Loop

[**Previous Page**](https://www.tutorialspoint.com/powershell/powershell_looping.htm)

[**Next Page**](https://www.tutorialspoint.com/powershell/powershell_looping.htm)

The following scripts demonstrates the while loop.

> $array = @("item1", "item2", "item3")

$counter = 0;

while($counter -lt $array.length){

$array[$counter]

$counter += 1

}

item1

item2

item3

Powershell - Do..While Loop

[**Previous Page**](https://www.tutorialspoint.com/powershell/powershell_looping.htm)

[**Next Page**](https://www.tutorialspoint.com/powershell/powershell_looping.htm)

The following scripts demonstrates the do.. while loop.

> $array = @("item1", "item2", "item3")

$counter = 0;

do {

$array[$counter]

$counter += 1

} while($counter -lt $array.length)

item1

item2

item3

Powershell - If Statement

[**Previous Page**](https://www.tutorialspoint.com/powershell/powershell_conditions.htm)

[**Next Page**](https://www.tutorialspoint.com/powershell/powershell_conditions.htm)

An **if** statement consists of a Boolean expression followed by one or more statements.

Syntax

Following is the syntax of an if statement −

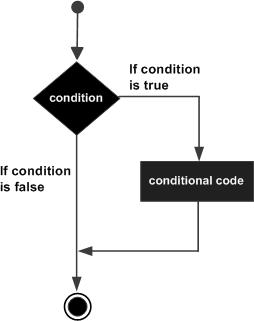
if(Boolean\_expression) {

// Statements will execute if the Boolean expression is true

}

If the Boolean expression evaluates to true then the block of code inside the if statement will be executed. If not, the first set of code after the end of the if statement (after the closing curly brace) will be executed.

Flow Diagram



Example

$x = 10

if($x -le 20){

write-host("This is if statement")

}

This will produce the following result −

Output

This is if statement.

Powershell - If Else Statement

[**Previous Page**](https://www.tutorialspoint.com/powershell/powershell_conditions.htm)

[**Next Page**](https://www.tutorialspoint.com/powershell/powershell_conditions.htm)

An **if** statement can be followed by an optional **else** statement, which executes when the Boolean expression is false.

Syntax

Following is the syntax of an if...else statement −

if(Boolean\_expression) {

// Executes when the Boolean expression is true

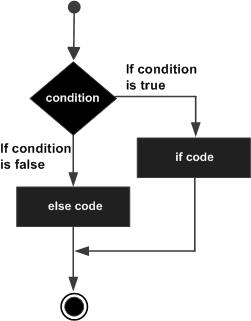
}else {

// Executes when the Boolean expression is false

}

If the boolean expression evaluates to true, then the if block of code will be executed, otherwise else block of code will be executed.

Flow Diagram



Example

$x = 30

if($x -le 20){

write-host("This is if statement")

}else {

write-host("This is else statement")

}

This will produce the following result −

Output

This is else statement

The if...elseif...else Statement

An if statement can be followed by an optional *else if...else* statement, which is very useful to test various conditions using single if...elseif statement.

When using if, elseif, else statements there are a few points to keep in mind.

* An if can have zero or one else's and it must come after any elseif's.
* An if can have zero to many elseif's and they must come before the else.
* Once an else if succeeds, none of the remaining elseif's or else's will be tested.

Syntax

Following is the syntax of an if...else statement −

if(Boolean\_expression 1) {

// Executes when the Boolean expression 1 is true

}elseif(Boolean\_expression 2) {

// Executes when the Boolean expression 2 is true

}elseif(Boolean\_expression 3) {

// Executes when the Boolean expression 3 is true

}else {

// Executes when the none of the above condition is true.

}

Example

$x = 30

if($x -eq 10){

write-host("Value of X is 10")

} elseif($x -eq 20){

write-host("Value of X is 20")

} elseif($x -eq 30){

write-host("Value of X is 30")

} else {

write-host("This is else statement")

}

This will produce the following result −

Output

Value of X is 30

Powershell - Nested If Else Statement

[**Previous Page**](https://www.tutorialspoint.com/powershell/powershell_conditions.htm)

[**Next Page**](https://www.tutorialspoint.com/powershell/powershell_conditions.htm)

It is always legal to nest if-else statements which means you can use one if or elseif statement inside another if or elseif statement.

Syntax

The syntax for a nested if...else is as follows −

if(Boolean\_expression 1) {

// Executes when the Boolean expression 1 is true

if(Boolean\_expression 2) {

// Executes when the Boolean expression 2 is true

}

}

You can nest **elseif...else** in the similar way as we have nested *if* statement.

Example

$x = 30

$y = 10

if($x -eq 30){

if($y -eq 10) {

write-host("X = 30 and Y = 10")

}

}

This will produce the following result −

Output

X = 30 and Y = 10

Powershell - Switch Statement

[**Previous Page**](https://www.tutorialspoint.com/powershell/powershell_conditions.htm)

[**Next Page**](https://www.tutorialspoint.com/powershell/powershell_conditions.htm)

A **switch** statement allows a variable to be tested for equality against a list of values. Each value is called a case, and the variable being switched on is checked for each case.

Syntax

The syntax of enhanced for loop is −

switch(<test-value>) {

<condition> {<action>}

break; // optional

<condition> {<action>}

break; // optional

<condition> {<action>}

break; // optional

}

The following rules apply to a **switch** statement −

* The variable used in a switch statement can only be any object or an array of objects.
* You can have any number of case statements within a switch. Each case is followed by optional action to be performed.
* The value for a case must be the same data type as the variable in the switch and it must be a constant or a literal.
* When the variable being switched on is equal to a case, the statements following that case will execute until a *break* statement is reached.
* When a *break* statement is reached, the switch terminates, and the flow of control jumps to the next line following the switch statement.
* Not every case needs to contain a break. If no break appears, the flow of control will *fall through*to subsequent cases until a break is reached.

Flow Diagram



Example 1

Example of Switch statment without break statment.

switch(3){

1 {"One"}

2 {"Two"}

3 {"Three"}

4 {"Four"}

3 {"Three Again"}

}

This will produce the following result −

Output

Three

Three Again

Example 2

Example of Switch statment with break statment.

switch(3){

1 {"One"}

2 {"Two"}

3 {"Three"; break }

4 {"Four"}

3 {"Three Again"}

}

This will produce the following result −

Output

Three

Example 3

Example of Switch statment with array as input.

switch(4,2){

1 {"One"}

2 {"Two"}

3 {"Three"; break }

4 {"Four"}

3 {"Three Again"}

}

This will produce the following result −

Output

Four

Two

# Powershell - Array

[**Previous Page**](https://www.tutorialspoint.com/powershell/powershell_conditions.htm)

[**Next Page**](https://www.tutorialspoint.com/powershell/powershell_hashtables.htm)

PowerShell provides a data structure, the **array**, which stores a fixed-size sequential collection of elements of the any type. An array is used to store a collection of data, but it is often more useful to think of an array as a collection of variables or objects.

Instead of declaring individual variables, such as number0, number1, ..., and number99, you declare one array variable such as numbers and use numbers[0], numbers[1], and ..., numbers[99] to represent individual variables.

This tutorial introduces how to declare array variables, create arrays, and process arrays using indexed variables.

## Declaring Array Variables

To use an array in a program, you must declare a variable to reference the array, and you can specify the type of array the variable can reference. Here is the syntax for declaring an array variable −

### **Syntax**

$A = 1, 2, 3, 4

or

$A = 1..4

**Note** − By default type of objects of array is System.Object. GetType() method returns the type of the array. Type can be passed.

### **Example**

The following code snippets are examples of this syntax −

[int32[]]$intA = 1500,2230,3350,4000

$A = 1, 2, 3, 4

$A.getType()

This will produce the following result −

## Output

IsPublic IsSerial Name BaseType

-------- -------- ---- --------

True True Object[] System.Array

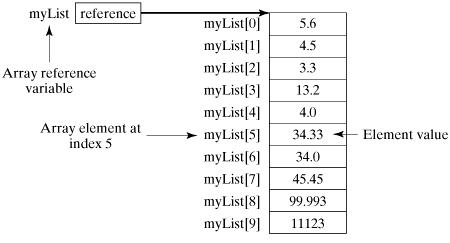
The array elements are accessed through the **index**. Array indices are 0-based; that is, they start from 0 to **arrayRefVar.length-1**.

### **Example**

Following statement declares an array variable, myList, creates an array of 10 elements of double type and assigns its reference to myList −

$myList = 5.6, 4.5, 3.3, 13.2, 4.0, 34.33, 34.0, 45.45, 99.993, 11123

Following picture represents array myList. Here, myList holds ten double values and the indices are from 0 to 9.



## Processing Arrays

When processing array elements, we often use either **for** loop or **foreach** loop because all of the elements in an array are of the same type and the size of the array is known.

### **Example**

Here is a complete example showing how to create, initialize, and process arrays −

$myList = 5.6, 4.5, 3.3, 13.2, 4.0, 34.33, 34.0, 45.45, 99.993, 11123

write-host("Print all the array elements")

$myList

write-host("Get the length of array")

$myList.Length

write-host("Get Second element of array")

$myList[1]

write-host("Get partial array")

$subList = $myList[1..3]

write-host("print subList")

$subList

write-host("using for loop")

for ($i = 0; $i -le ($myList.length - 1); $i += 1) {

$myList[$i]

}

write-host("using forEach Loop")

foreach ($element in $myList) {

$element

}

write-host("using while Loop")

$i = 0

while($i -lt 4) {

$myList[$i];

$i++

}

write-host("Assign values")

$myList[1] = 10

$myList

This will produce the following result −

### **Output**

Print all the array elements

5.6

4.5

3.3

13.2

4

34.33

34

45.45

99.993

11123

Get the length of array

10

Get Second element of array

4.5

Get partial array

print subList

4.5

3.3

13.2

using for loop

5.6

4.5

3.3

13.2

4

34.33

34

45.45

99.993

11123

using forEach Loop

5.6

4.5

3.3

13.2

4

34.33

34

45.45

99.993

11123

using while Loop

5.6

4.5

3.3

13.2

Assign values

5.6

10

3.3

13.2

4

34.33

34

45.45

99.993

11123

## The Arrays Methods Examples

Here is a complete example showing operations on arrays using its methods

$myList = @(0..4)

write-host("Print array")

$myList

$myList = @(0..4)

write-host("Assign values")

$myList[1] = 10

$myList

This will produce the following result −

### **Output**

Clear array

Print array

0

1

2

3

4

Assign values

0

10

2

3

4

# Powershell - Hashtables

[**Previous Page**](https://www.tutorialspoint.com/powershell/powershell_array.htm)

[**Next Page**](https://www.tutorialspoint.com/powershell/powershell_regex.htm)

Hashtable stores key/value pairs in a hash table. When using a Hashtable, you specify an object that is used as a key, and the value that you want linked to that key. Generally we used String or numbers as keys.

This tutorial introduces how to declare hashtable variables, create hashtables, and process hashtable using its methods.

## Declaring hashtable Variables

To use an hashtable in a program, you must declare a variable to reference the hashtable. Here is the syntax for declaring an hashtable variable −

### **Syntax**

$hash = @{ ID = 1; Shape = "Square"; Color = "Blue"}

or

$hash = @{}

**Note** − Ordered dictionaries can be created using similar syntax. Ordered dictionaries maintain the order in which entries are added whereas hashtables do not.

### **Example**

The following code snippets are examples of this syntax −

$hash = [ordered]@{ ID = 1; Shape = "Square"; Color = "Blue"}

Print the hashtable.

$hash

### **Output**

Name Value

---- -----

ID 1

Color Blue

Shape Square

The hashtable values are accessed through the **keys**.

> $hash["ID"]

1

## Processing Hashtable

Dot notation can be used to access hashtables keys or values.

> $hash.keys

ID

Color

Shape

> $hash.values

1

Blue

Square

### **Example**

Here is a complete example showing how to create, initialize, and process hashtable −

$hash = @{ ID = 1; Shape = "Square"; Color = "Blue"}

write-host("Print all hashtable keys")

$hash.keys

write-host("Print all hashtable values")

$hash.values

write-host("Get ID")

$hash["ID"]

write-host("Get Shape")

$hash.Number

write-host("print Size")

$hash.Count

write-host("Add key-value")

$hash["Updated"] = "Now"

write-host("Add key-value")

$hash.Add("Created","Now")

write-host("print Size")

$hash.Count

write-host("Remove key-value")

$hash.Remove("Updated")

write-host("print Size")

$hash.Count

write-host("sort by key")

$hash.GetEnumerator() | Sort-Object -Property key

This will produce the following result −

### **Output**

Print all hashtable keys

ID

Color

Shape

Print all hashtable values

1

Blue

Square

Get ID

1

Get Shape

print Size

3

Add key-value

Add key-value

print Size

5

Remove key-value

print Size

4

sort by key

Name Value

---- -----

Color Blue

Created Now

ID 1

Shape

Square

***1.) Create a folder TestingPurpose and 3 Subfolders inside it SubFolder1, SubFolder2***

***2.) Create some test files inside these folders:***

***TypeATest1.txt, TypeATest2.txt  … TypeATest50.txt into SubFolder1***

***TypeBTest51.txt, Purpose52Test2.txt … TypeBTest100 into SubFolder2***

***Needless to say that you have to use logic for creating these files. Not one by one***

***3.) Move all files which have an odd number in its name to SubFolder2***

***4.) Move all files which have even number in its name to SubFolder1***

***5.) Rename folder SubFolder1 to EvenFilesContainer and SubFolder2 to OddFilesContainer***

***5.) Prepare a list of all files currently existing inside folder TestingPurpose***

***Example: MasterFile.txt:***

***As of YYYYMMDD HH: MM files inside Testing Purpose are:***

***C:\testingPurpose\EvenFilesContainer\TypeBTest2.txt***

***.***

***.***

***C:\testingPurpose\OddFilesContainer\TypeATest99.txt***

***6.) Delete all files which start with TypeA***