Windows PowerShell Get-Help on Cmdlet 'New-Variable'

PS:\>Get-HELP New-Variable -Full

NAME

New-Variable

SYNOPSIS

Creates a new variable.

SYNTAX

New-Variable [-Name] <System.String> [[-Value] <System.Object>] [-Description <System.String>] [-Force] [-Option {None | ReadOnly | Constant | Private | AllScope |

Unspecified}] [-PassThru] [-Scope <System.String>] [-Visibility {Public | Private}] [-Confirm] [-WhatIf] [<CommonParameters>]

DESCRIPTION

The `New-Variable` cmdlet creates a new variable in PowerShell. You can assign a value to the variable while creating it or assign or change the value after it is

created.

You can use the parameters of `New-Variable` to set the properties of the variable, set the scope of a variable, and

determine whether variables are public or private.

Typically, you create a new variable by typing the variable name and its value, such as `$Var = 3`, but you can use the `New-Variable` cmdlet to use its parameters.

PARAMETERS

-Description <System.String>

Specifies a description of the variable.

Required? false
Position? named
Default value None
Accept pipeline input? False
Accept wildcard characters? false

-Force <System.Management.Automation.SwitchParameter>

Indicates that the cmdlet creates a variable with the same name as an existing read-only variable.

By default, you can overwrite a variable unless the variable has an option value of `ReadOnly` or `Constant`. For more information, see the Option parameter.

Required? false
Position? named
Default value False
Accept pipeline input? False
Accept wildcard characters? false

-Name <System.String>

Specifies a name for the new variable.

Required? true
Position? 0

Default value None

Accept pipeline input? True (ByPropertyName)

Accept wildcard characters? false

-Option <System.Management.Automation.ScopedItemOptions>

Specifies the value of the Options property of the variable. The acceptable values for this parameter are:

- `None` - Sets no options. `None` is the default.

- `ReadOnly` - Can be deleted. Cannot be changed, except by using the Force parameter. - `Private` - The variable is available only in the current scope.

- `AllScope` - The variable is copied to any new scopes that are created.

- `Constant` - Cannot be deleted or changed. `Constant` is valid only when you are creating a

variable. You cannot change the options of an existing variable to `Constant`.

These values are defined as a flag-based enumeration. You can combine multiple values together to set multiple flags using this parameter. The values can be

passed to the Option parameter as an array of values or as a comma-separated string of those values. The cmdlet will combine the values using a binary-OR

operation. Passing values as an array is the simplest option and also allows you to use tab-completion on the values.

To see the Options property of all variables in the session, type `Get-Variable | Format-Table -Property name, options -AutoSize`.

Required? false

Position? named

Default value None

Accept pipeline input? False

Accept wildcard characters? false

-PassThru <System.Management.Automation.SwitchParameter>

Returns an object representing the item with which you are working. By default, this cmdlet does not generate any output.

Required? false

Position? named

Default value False

Accept pipeline input? False

Accept wildcard characters? false

-Scope <System.String>

Specifies the scope of the new variable. The acceptable values for this parameter are:

- `Global` - Variables created in the global scope are accessible everywhere in a PowerShell process. - `Local` - The local scope refers to the current scope,

this can be any scope depending on the context. `Local` is the default scope when the scope parameter is not specified. - `Script` - Variables created in the

script scope are accessible only within the script file or module they are created in. - A number relative to the current scope (0 through the number of scopes,

where 0 is the current scope, 1 is its parent, 2 the parent of the parent scope, and so on). Negative numbers cannot be used.

> [!NOTE] > The parameter also accepts the value of `Private`. `Private` is not actually a scope but an > optional setting for a variable. However, using the

`Private` value with this cmdlet does not > change the visibility of the variable. For more information, see > about_Scopes

(../Microsoft.PowerShell.Core/About/about_Scopes.md).

Required? false

Position? named

Default value None

Accept pipeline input? False

Accept wildcard characters? false

-Value <System.Object>

Specifies the initial value of the variable.

Required? false

Position? 1

Default value None

Accept pipeline input? True (ByPropertyName, ByValue)

Accept wildcard characters? false

-Visibility <System.Management.Automation.SessionStateEntryVisibility>

Determines whether the variable is visible outside of the session in which it was created. This parameter is designed for use in scripts and commands that will be

delivered to other users. The acceptable values for this parameter are:

- `Public` - The variable is visible. `Public` is the default.

- `Private` - The variable is not visible.

When a variable is private, it does not appear in lists of variables, such as those returned by `Get-Variable`, or in displays of the `Variable:` drive. Commands

to read or change the value of a private variable return an error. However, the user can run commands that use a private variable if the commands were written in

the session in which the variable was defined.

Required? false

Position? named

Default value None

Accept pipeline input? False

Accept wildcard characters? false

-Confirm <System.Management.Automation.SwitchParameter>

Prompts you for confirmation before running the cmdlet.

Required? false

Position? named

Default value False

Accept pipeline input? False

Accept wildcard characters? false

-WhatIf <System.Management.Automation.SwitchParameter>

Shows what would happen if the cmdlet runs. The cmdlet is not run.

Required? false

Position? named

Default value False

Accept pipeline input? False

Accept wildcard characters? false

<CommonParameters>

This cmdlet supports the common parameters: Verbose, Debug,

ErrorAction, ErrorVariable, WarningAction, WarningVariable,

OutBuffer, PipelineVariable, and OutVariable. For more information, see

about_CommonParameters (https:/go.microsoft.com/fwlink/?LinkID=113216).

INPUTS

System.Object

You can pipe any object to `New-Variable`.

OUTPUTS

None

By default, this cmdlet returns no output.

System.Management.Automation.PSVariable

When you use the PassThru parameter, this cmdlet returns a PSVariable object representing the new variable.

NOTES

Windows PowerShell includes the following aliases for `New-Variable`:

- `nv`

---------- Example 1: Create a variable ----------

New-Variable days

This command creates a new variable named days. You are not required to type the Name parameter.

------ Example 2: Create a variable and assign it a value ------

New-Variable -Name "zipcode" -Value 98033

This command creates a variable named zipcode and assigns it the value 98033.

---- Example 3: Create a variable with the ReadOnly option ----

PS C:\> New-Variable -Name Max -Value 256 -Option ReadOnly

PS C:\> New-Variable -Name max -Value 1024

New-Variable : A variable with name 'max' already exists.

At line:1 char:1

+ New-Variable -Name max -Value 1024

+ ~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~

+ CategoryInfo : ResourceExists: (max:String) [New-Variable], SessionStateException

+ FullyQualifiedErrorId : VariableAlreadyExists,Microsoft.PowerShell.Commands.NewVariableCommand

PS C:\> New-Variable -Name max -Value 1024 -Force

This example shows how to use the `ReadOnly` option of `New-Variable` to protect a variable from being overwritten.

The first command creates a new variable named Max and sets its value to 256. It uses the Option parameter with a value of `ReadOnly`.

The second command tries to create a second variable with the same name. This command returns an error, because the read-only option is set on the variable.

The third command uses the Force parameter to override the read-only protection on the variable. In this case, the command to create a new variable with the same name

succeeds.

------- Example 4: Assign multiple options to a variable -------

New-Variable -Name 'TestVariable' -Value 'Test Value' -Option AllScope,Constant

This example creates a variable and assigns the `AllScope` and `Constant` options so the variable will be available in the current scope and any new scopes created

and cannot be changed or deleted.

---------- Example 5: Create a private variable ----------

PS C:\> New-Variable -Name counter -Visibility Private

#Effect of private variable in a module.

PS C:\> Get-Variable c*

Name Value

---- -----

Culture en-US

ConsoleFileName

ConfirmPreference High

```
CommandLineParameters          {}

PS C:\> $counter
"Cannot access the variable '$counter' because it is a private variable"
At line:1 char:1
+ $counter
+ ~~~~~~~~
    + CategoryInfo          : PermissionDenied: (counter:String) [], SessionStateException
    + FullyQualifiedErrorId : VariableIsPrivate

PS C:\> Get-Counter
Name         Value
----         -----
Counter1     3.1415
...
```

The sample output shows the behavior of a private variable. The user who has loaded the module cannot view or change the value of the Counter variable, but the Counter variable can be read and changed by the commands in the module.

```
---------- Example 6: Create a variable with a space ----------

PS C:\> New-Variable -Name 'with space' -Value 'abc123xyz'

PS C:\> Get-Variable -Name 'with space'

Name                           Value
----                           -----
with space                     abc123xyz

PS C:\> ${with space}
abc123xyz
```

RELATED LINKS

Online Version:

https://learn.microsoft.com/powershell/module/microsoft.powershell.utility/new-variable?view=powershell-5.1&WT.mc_id=ps-gethelp

Clear-Variable

Get-Variable

Remove-Variable

Set-Variable