Using variables to store objects

08/27/20182 minutes to read  

In this article

PowerShell works with objects. PowerShell lets you create named objects known as variables. Variable names can include the underscore character and any alphanumeric characters. When used in PowerShell, a variable is always specified using the $ character followed by variable name.

Creating a variable

You can create a variable by typing a valid variable name:

Copy

```
PS> $loc
PS>
```

This example returns no result because `$loc` doesn't have a value. You can create a variable and assign it a value in the same step. PowerShell only creates the variable if it doesn't exist. Otherwise, it assigns the specified value to the existing variable. The following example stores the current location in the variable `$loc`:

Copy

PowerShell Copy

```
$loc = Get-Location
```

PowerShell displays no output when you type this command. PowerShell sends the output of 'Get-Location' to `$loc`. In PowerShell, data that isn't assigned or redirected is sent to the screen. Typing `$loc` shows your current location:

Copy

```
PS> $loc

Path
----
C:\temp
```

You can use `Get-Member` to display information about the contents of variables. `Get-Member` shows you that `$loc` is a **PathInfo** object, just like the output from `Get-Location`:

PowerShell Copy

```
PS> $loc | Get-Member -MemberType Property

   TypeName: System.Management.Automation.-
PathInfo

Name         MemberType Definition
----         ---------- ----------
Drive        Property   System.Management.Au-
tomation.PSDriveInfo Drive {get;}
Path         Property   System.String Path
{get;}
Provider     Property   System.Management.Au-
```

```
tomation.ProviderInfo Provider {...
ProviderPath Property   System.String
ProviderPath {get;}
```

Manipulating variables

PowerShell provides several commands to manipulate variables. You can see a complete listing in a readable form by typing:

PowerShell

Copy

```
Get-Command -Noun Variable | Format-Table -
Property Name,Definition -AutoSize -Wrap
```

PowerShell also creates several system-defined variables. You can use the `Remove-Variable` cmdlet to remove variables, which are not controlled by PowerShell, from the current session. Type the following command to clear all variables:

PowerShell

Copy

```
Remove-Variable -Name * -Force -ErrorAction
SilentlyContinue
```

After running the previous command, the `Get-Variable` cmdlet shows the PowerShell system variables.

PowerShell also creates a variable drive. Use the following example to display all PowerShell variables using the variable drive:

PowerShell

Copy

```
Get-ChildItem variable:
```

Using cmd.exe variables

PowerShell can use the same environment variables available to any Windows process, including **cmd.exe**. These variables are exposed through a drive named `env:`. You can view these variables by typing the following command:

PowerShell

Copy

```
Get-ChildItem env:
```

The standard `*-Variable` cmdlets aren't designed to work with environment variables. Environment variables are accessed using the `env:` drive prefix. For example, the **%SystemRoot%** variable in **cmd.exe**contains the operating system's root directory name. In PowerShell, you use `$env:SystemRoot` to access the same value.

Copy

```
PS> $env:SystemRoot
C:\WINDOWS
```

You can also create and modify environment variables from within PowerShell. Environment variables in PowerShell follow the same rules for environment variables used elsewhere in the operating system. The following example creates a new environment variable:

PowerShell

Copy

```
$env:LIB_PATH='/usr/local/lib'
```

Though not required, it's common for environment variable names to use all uppercase letters.