

Working with variables

To create a new variable, use an assignment statement to assign a value to the variable.

```
$MyVariable = 1, 2, 3
$Path = "C:\Windows\System32"
```

To display the value of a variable, type the variable name, preceded by a dollar sign (\$).

```
$MyVariable
1
2
3
```

To change the value of a variable, assign a new value to the variable.

```
$MyVariable = "The green cat."
$MyVariable
```

To delete the value of a variable, use the Clear-Variable cmdlet or change the value to \$null.

```
Clear-Variable -Name MyVariable
$MyVariable = $null
```

To delete the variable, use Remove-Variable or Remove-Item.

```
Remove-Variable -Name MyVariable
Remove-Item -Path Variable:\MyVariable
```

To get a list of all the variables in your PowerShell session, type Get-Variable.

Variables are useful for storing the results of commands.

```
$Processes = Get-Process
$Today = (Get-Date).DateTime
```

It is also possible to assign values to multiple variables with one statement.

```
$a = $b = $c = 0
```

The next example assigns multiple values to multiple variables.

```
$i,$j,$k = 10, "red", $true
# $i is 10, $j is "red", $k is True
$i,$j = 10, "red", $true
# $i is 10, $j is [object[]], Length 2
```

Types of variables

```
$a = 12
System.Int32

$a = "Word"
System.String

$a = 12, "Word"
array of System.Int32, System.String

$a = Get-ChildItem C:\Windows
FileInfo and DirectoryInfo types

To use cast notation, enter a type name, enclosed in brackets, before the variable name (on the left side of the assignment statement).
[int]$number = 8
```

Variable substitution in strings

```
Concatenation
$name = 'Kevin Marquette'
$message = 'Hello, ' + $name

Variable substitution
$first = 'Kevin'
$last = 'Marquette'
$message = "Hello, $first $last."
```

Arrays

To create and initialize an array, assign multiple values to a variable.

```
$A = 22,5,10,8,12,9,80
```

The array sub-expression operator creates an array from the statements inside it.

```
@( ... )
$a = @("Hello World")
$p = @(Get-Process Notepad)
```

```
Where-Object filtering
$data | Where-Object {$_.FirstName -eq 'Kevin'}
$data | Where FirstName -eq Kevin
```

```
Where()
$data.Where({$_ .FirstName -eq 'Kevin'})
```

```
Selects objects or object properties.
Get-Process | Select-Object -Property ProcessName, Id, WS
```

Hash Tables

To create an empty hashtable in the value of \$hash, type:

```
$hash = @{ }
```

You can also add keys and values to a hashtable when you create it.

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

Hash Tables (cont)

To display a hashtable that's saved in a variable, type the variable name.	<code>\$hash</code>
hashtables have Keys and Values properties.	<code>\$hash.keys</code> <code>\$hash.values</code>
You can iterate over the keys in a hashtable to process the values in several ways.	<pre>foreach (\$Key in \$hash.Keys) { "The value of '\$Key' is: \$(\$hash[\$Key])" }</pre>
To add keys and values to a hashtable, use the following command format.	<code>\$hash["<key>"] = "<value>"</code> <code>\$hash["Time"] = "Now"</code>
You can also add keys and values to a hashtable using the Add method of the System.Collections.Hashtable object.	<code>Add(Key, Value)</code> <code>\$hash.Add("Time", "Now")</code>

PSCustomObject

Creating a PSCustomObject	<code>\$myObject = [PSCustomObject]@{ Name = 'Kevin' Language = 'PowerShell' State = 'Texas' }</code>
Converting a hashtable	<code>\$myHashtable = @{ Name = 'Kevin' Language = 'PowerShell' State = 'Texas' }</code> <code>\$myObject = [pscustomobject]\$myHashtable</code>
Saving to a file	<code>\$myObject ConvertTo-Json -depth 1 Set-Content -Path \$Path</code> <code>\$myObject = Get-Content -Path \$Path ConvertFrom-Json</code>
Adding properties	<code>\$myObject Add-Member -MemberType NoteProperty -Name 'ID' -Value 'KevinMarquette'</code> <code>\$myObject.ID</code>
Remove properties	<code>\$myObject.psobject.properties.remove('ID')</code>

PSCustomObject (cont)

Enumerating property names	<code>\$myObject Get-Member -MemberType NoteProperty Select -ExpandProperty Name</code>
Dynamically accessing properties	<code>\$myObject.'Name'</code>
Convert PSCustomObject into a hashtable	<code>\$hashtable = @{} foreach(\$property in \$myobject.psobject.properties.name) { \$hashtable[\$property] = \$myObject.\$property }</code>
Testing for properties	<code>if(\$null -ne \$myObject.ID) if(\$myobject.psobject.properties.match('ID').Count)</code>

Functions

A simple function	<code>function Get-Version { \$PSVersionTable.PSVersion }</code>
Parameters	<code>function Test-MrParameter { param (\$ComputerName) Write-Output \$ComputerName }</code>