PowerShell Get-Help on command 'Mock'

PS C:\Users\wahid> Get-Help Mock

NAME

Mock

SYNOPSIS

Mocks the behavior of an existing command with an alternate implementation.

SYNTAX

```
Mock [[-CommandName] <String>] [[-MockWith] <ScriptBlock>] [-Verifiable]
[[-ParameterFilter] <ScriptBlock>] [[-ModuleName] <String>]
[<CommonParameters>]
```

DESCRIPTION

This creates new behavior for any existing command within the scope of a Describe or Context block. The function allows you to specify a script block that will become the command's new behavior.

Optionally, you may create a Parameter Filter which will examine the parameters passed to the mocked command and will invoke the mocked

behavior only if the values of the parameter values pass the filter. If
they do not, the original command implementation will be invoked instead
of a mock.

You may create multiple mocks for the same command, each using a different
ParameterFilter. ParameterFilters will be evaluated in reverse order of
their creation. The last one created will be the first to be evaluated.
The mock of the first filter to pass will be used. The exception to this
rule are Mocks with no filters. They will always be evaluated last since
they will act as a "catch all" mock.

Mocks can be marked Verifiable. If so, the Assert-VerifiableMocks command
can be used to check if all Verifiable mocks were actually called. If any
verifiable mock is not called, Assert-VerifiableMocks will throw an
exception and indicate all mocks not called.

If you wish to mock commands that are called from inside a script module,
you can do so by using the -ModuleName parameter to the Mock command. This
injects the mock into the specified module. If you do not specify a
module name, the mock will be created in the same scope as the test script.
You may mock the same command multiple times, in different scopes, as needed.
Each module's mock maintains a separate call history and verified status.

PARAMETERS

-CommandName <String>

The name of the command to be mocked.

-MockWith <ScriptBlock>

A ScriptBlock specifying the behvior that will be used to mock CommandName.
The default is an empty ScriptBlock.
NOTE: Do not specify param or dynamicparam blocks in this script block.
These will be injected automatically based on the signature of the command

being mocked, and the MockWith script block can contain references to the
mocked commands parameter variables.

-Verifiable [<SwitchParameter>]
When this is set, the mock will be checked when Assert-VerifiableMocks is
called.

-ParameterFilter <ScriptBlock>
An optional filter to limit mocking behavior only to usages of
CommandName where the values of the parameters passed to the command
pass the filter.

This ScriptBlock must return a boolean value. See examples for usage.

-ModuleName <String>
Optional string specifying the name of the module where this command
is to be mocked. This should be a module that _calls_ the mocked
command; it doesn't necessarily have to be the same module which
originally implemented the command.

<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).

-------------------------- EXAMPLE 1 --------------------------

```
PS C:\>Mock Get-ChildItem { return @{FullName = "A_File.TXT"} }
```

Using this Mock, all calls to Get-ChildItem will return a hashtable with a
FullName property returning "A_File.TXT"

-------------------------- EXAMPLE 2 --------------------------

```
PS C:\>Mock Get-ChildItem { return @{FullName = "A_File.TXT"} }
-ParameterFilter { $Path -and $Path.StartsWith($env:temp) }
```

This Mock will only be applied to Get-ChildItem calls within the user's temp directory.

-------------------------- EXAMPLE 3 --------------------------

```
PS C:\>Mock Set-Content {} -Verifiable -ParameterFilter { $Path -eq
"some_path" -and $Value -eq "Expected Value" }
```

When this mock is used, if the Mock is never invoked and Assert-VerifiableMocks is called, an exception will be thrown. The command behavior will do nothing since the ScriptBlock is empty.

-------------------------- EXAMPLE 4 --------------------------

```
PS C:\>Mock Get-ChildItem { return @{FullName = "A_File.TXT"} }
-ParameterFilter { $Path -and $Path.StartsWith($env:temp\1) }

Mock Get-ChildItem { return @{FullName = "B_File.TXT"} } -ParameterFilter {
$Path -and $Path.StartsWith($env:temp\2) }
```

```
Mock Get-ChildItem { return @{FullName = "C_File.TXT"} } -ParameterFilter {
$Path -and $Path.StartsWith($env:temp\3) }
```

Multiple mocks of the same command may be used. The parameter filter
determines which is invoked. Here, if Get-ChildItem is called on the "2"
directory of the temp folder, then B_File.txt will be returned.

-------------------------- EXAMPLE 5 --------------------------

```
PS C:\>Mock Get-ChildItem { return @{FullName="B_File.TXT"} } -ParameterFilter
{ $Path -eq "$env:temp\me" }

Mock Get-ChildItem { return @{FullName="A_File.TXT"} } -ParameterFilter {
$Path -and $Path.StartsWith($env:temp) }

Get-ChildItem $env:temp\me
```

Here, both mocks could apply since both filters will pass. A_File.TXT will be
returned because it was the most recent Mock created.

-------------------------- EXAMPLE 6 --------------------------

```
PS C:\>Mock Get-ChildItem { return @{FullName = "B_File.TXT"} }
-ParameterFilter { $Path -eq "$env:temp\me" }

Mock Get-ChildItem { return @{FullName = "A_File.TXT"} }
```

Get-ChildItem c:\windows

Here, A_File.TXT will be returned. Since no filter was specified, it will

apply to any call to Get-ChildItem that does not pass another filter.

-------------------------- EXAMPLE 7 --------------------------

```
PS C:\>Mock Get-ChildItem { return @{FullName = "B_File.TXT"} }
-ParameterFilter { $Path -eq "$env:temp\me" }

Mock Get-ChildItem { return @{FullName = "A_File.TXT"} }

Get-ChildItem $env:temp\me
```

Here, B_File.TXT will be returned. Even though the filterless mock was created

more recently. This illustrates that filterless Mocks are always evaluated

last regardlss of their creation order.

-------------------------- EXAMPLE 8 --------------------------

```
PS C:\>Mock Get-ChildItem { return @{FullName = "A_File.TXT"} } -ModuleName
MyTestModule
```

Using this Mock, all calls to Get-ChildItem from within the MyTestModule module

will return a hashtable with a FullName property returning "A_File.TXT"

```
-------------------------- EXAMPLE 9 --------------------------

PS C:\>Get-Module -Name ModuleMockExample | Remove-Module

New-Module -Name ModuleMockExample  -ScriptBlock {
    function Hidden { "Internal Module Function" }
    function Exported { Hidden }

    Export-ModuleMember -Function Exported
} | Import-Module -Force

Describe "ModuleMockExample" {

    It "Hidden function is not directly accessible outside the module" {
        { Hidden } | Should Throw
    }

    It "Original Hidden function is called" {
        Exported | Should Be "Internal Module Function"
    }

    It "Hidden is replaced with our implementation" {
        Mock Hidden { "Mocked" } -ModuleName ModuleMockExample
        Exported | Should Be "Mocked"
    }
}
```

This example shows how calls to commands made from inside a module can be mocked by using the -ModuleName parameter.

REMARKS

To see the examples, type: "get-help Mock -examples".

For more information, type: "get-help Mock -detailed".

For technical information, type: "get-help Mock -full".

For online help, type: "get-help Mock -online"