Full credit is given to the above companies including the OS that this PDF file was generated!

PowerShell Get-Help on command 'Invoke-Pester'

PS C:\Users\wahid> Get-Help Invoke-Pester

NAME

Invoke-Pester

SYNOPSIS

Invokes Pester to run all tests (files containing *.Tests.ps1) recursively under the Path

SYNTAX

Invoke-Pester [[-Script] <Object[]>] [[-TestName] <String[]>] [[-EnableExit]] [[-OutputXml] <String>] [[-Tag] <String[]>] [-ExcludeTag <String[]>] [-PassThru] [-CodeCoverage <Object[]>] [-Strict] [-Quiet] [-PesterOption <Object>] [<CommonParameters>]

Invoke-Pester [[-Script] <Object[]>] [[-TestName] <String[]>] [[-EnableExit]] [[-Tag] <String[]>] [-ExcludeTag <String[]>] [-PassThru] [-CodeCoverage <Object[]>] [-Strict] -OutputFile <String> -OutputFormat <String> [-Quiet] [-PesterOption <Object>] [<CommonParameters>]

DESCRIPTION

Upon calling Invoke-Pester, all files that have a name containing
"*.Tests.ps1" will have the tests defined in their Describe blocks
executed. Invoke-Pester begins at the location of Path and
runs recursively through each sub directory looking for
"*.Tests.ps1" files containing tests. If a TestName is provided,
Invoke-Pester will only run tests that have a describe block with a
matching name. By default, Invoke-Pester will end the test run with a
simple report of the number of tests passed and failed output to the
console. One may want pester to "fail a build" in the event that any
tests fail. To accomodate this, Invoke-Pester will return an exit
code equal to the number of failed tests if the EnableExit switch is
set. Invoke-Pester will also write a NUnit style log of test results
if the OutputXml parameter is provided. In these cases, Invoke-Pester
will write the result log to the path provided in the OutputXml
parameter.

Optionally, Pester can generate a report of how much code is covered
by the tests, and information about any commands which were not
executed.

PARAMETERS

-Script <Object[]>

This parameter indicates which test scripts should be run.
This parameter may be passed simple strings (wildcards are allowed), or
hashtables containing Path, Arguments and Parameters keys.
If hashtables are used, the Parameters key must refer to a hashtable, and
the Arguments key must refer to an array; these will be splatted to the
test script(s) indicated in the Path key.

Note: If the path contains any wildcards, or if it refers to a directory,
then Pester will search for and execute all test scripts named *.Tests.ps1
in the target path; the search is recursive. If the path contains no

wildcards and refers to a file, Pester will just try to execute that file
regardless of its name.

Aliased to 'Path' and 'relative_path' for backwards compatibility.

-TestName <String[]>
Informs Invoke-Pester to only run Describe blocks that match this name.

-EnableExit [<SwitchParameter>]
Will cause Invoke-Pester to exit with a exit code equal to the number of
failed tests once all tests have been run. Use this to "fail" a build when
any tests fail.

-OutputXml <String>
The path where Invoke-Pester will save a NUnit formatted test results log
file. If this path is not provided, no log will be generated.

-Tag <String[]>
Informs Invoke-Pester to only run Describe blocks tagged with the tags
specified. Aliased 'Tags' for backwards compatibility.

-ExcludeTag <String[]>
Informs Invoke-Pester to not run blocks tagged with the tags specified.

-PassThru [<SwitchParameter>]
Returns a Pester result object containing the information about the whole
test run, and each test.

-CodeCoverage <Object[]>
Instructs Pester to generate a code coverage report in addition to running
tests. You may pass either hashtables or strings to this parameter.
If strings are used, they must be paths (wildcards allowed) to source
files, and all commands in the files are analyzed for code coverage.

By passing hashtables instead, you can limit the analysis to specific
lines or functions within a file.
Hashtables must contain a Path key (which can be abbreviated to just "P"),
and may contain Function (or "F"), StartLine (or "S"), and EndLine ("E")
keys to narrow down the commands to be analyzed.
If Function is specified, StartLine and EndLine are ignored.
If only StartLine is defined, the entire script file starting with
StartLine is analyzed.
If only EndLine is present, all lines in the script file up to and
including EndLine are analyzed.
Both Function and Path (as well as simple strings passed instead of
hashtables) may contain wildcards.

-Strict [<SwitchParameter>]
Makes Pending and Skipped tests to Failed tests. Useful for continuous
integration where you need to make sure all tests passed.

-OutputFile <String>

-OutputFormat <String>

-Quiet [<SwitchParameter>]
Disables the output Pester writes to screen. No other output is generated
unless you specify PassThru, or one of the Output parameters.

-PesterOption <Object>
Sets advanced options for the test execution. Enter a PesterOption object,
such as one that you create by using the New-PesterOption cmdlet, or a
hash table in which the keys are option names and the values are option
values.
For more information on the options available, see the help for
New-PesterOption.

<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:\>Invoke-Pester

This will find all *.Tests.ps1 files and run their tests. No exit code will be
returned and no log file will be saved.

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

PS C:\>Invoke-Pester -Script ./tests/Utils*

This will run all tests in files under ./Tests that begin with Utils and
alsocontains .Tests.

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

PS C:\>Invoke-Pester -Script @{ Path = './tests/Utils*'; Parameters = @{
NamedParameter = 'Passed By Name' }; Arguments = @('Passed by position') }

Executes the same tests as in Example 1, but will run them with the equivalent
of the following command line: & $testScriptPath -NamedParameter 'Passed By

Name' 'Passed by position'

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

PS C:\>Invoke-Pester -TestName "Add Numbers"

This will only run the Describe block named "Add Numbers"

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

PS C:\>Invoke-Pester -EnableExit -OutputXml "./artifacts/TestResults.xml"

This runs all tests from the current directory downwards and writes the
results according to the NUnit schema to artifacts/TestResults.xml just below
the current directory. The test run will return an exit code equal to the
number of test failures.

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

PS C:\>Invoke-Pester -EnableExit -OutputFile "./artifacts/TestResults.xml"
-OutputFormat NUnitxml

This runs all tests from the current directory downwards and writes the
results to an output file and NUnitxml output format

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

PS C:\>Invoke-Pester -CodeCoverage 'ScriptUnderTest.ps1'

Runs all *.Tests.ps1 scripts in the current directory, and generates a
coverage report for all commands in the "ScriptUnderTest.ps1" file.

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

PS C:\>Invoke-Pester -CodeCoverage @{ Path = 'ScriptUnderTest.ps1'; Function =
'FunctionUnderTest' }

Runs all *.Tests.ps1 scripts in the current directory, and generates a
coverage report for all commands in the "FunctionUnderTest" function in the
"ScriptUnderTest.ps1" file.

---------- EXAMPLE 9 ----------

PS C:\>Invoke-Pester -CodeCoverage @{ Path = 'ScriptUnderTest.ps1'; StartLine
= 10; EndLine = 20 }

Runs all *.Tests.ps1 scripts in the current directory, and generates a
coverage report for all commands on lines 10 through 20 in the

"ScriptUnderTest.ps1" file.

REMARKS

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

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

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

For online help, type: "get-help Invoke-Pester -online"