PowerShell Get-Help on command 'Select-String'

PS C:\Users\wahid> Get-Help Select-String

```
NAME
    Select-String


SYNOPSIS
    Finds text in strings and files.



SYNTAX
    Select-String [-Pattern] <System.String[]> [-AllMatches] [-CaseSensitive]
    [-Context <System.Int32[]>] [-Encoding {ASCII | BigEndianUnicode | Default |
    OEM | Unicode | UTF7 | UTF8 | UTF32}] [-Exclude <System.String[]>] [-Include
    <System.String[]>] -InputObject <System.Management.Automation.PSObject>
    [-List] [-NotMatch] [-Quiet] [-SimpleMatch] [<CommonParameters>]

    Select-String [-Pattern] <System.String[]> [-AllMatches] [-CaseSensitive]
    [-Context <System.Int32[]>] [-Encoding {ASCII | BigEndianUnicode | Default |
    OEM | Unicode | UTF7 | UTF8 | UTF32}] [-Exclude <System.String[]>] [-Include
    <System.String[]>] [-List] -LiteralPath <System.String[]> [-NotMatch] [-Quiet]
    [-SimpleMatch] [<CommonParameters>]

    Select-String [-Pattern] <System.String[]> [-Path] <System.String[]>
```

[-AllMatches] [-CaseSensitive] [-Context <System.Int32[]>] [-Encoding {ASCII |
BigEndianUnicode | Default | OEM | Unicode | UTF7 | UTF8 | UTF32}] [-Exclude
<System.String[]>] [-Include <System.String[]>] [-List] [-NotMatch] [-Quiet]
[-SimpleMatch] [<CommonParameters>]

DESCRIPTION

The `Select-String` cmdlet uses regular expression matching to search for text patterns in input strings and files. You can use `Select-String` similar to `grep` in UNIX or `findstr.exe` in Windows.

`Select-String` is based on lines of text. By default, `Select-String` finds the first match in each line and, for each match, it displays the file name, line number, and all text in the line containing the match. You can direct `Select-String` to find multiple matches per line, display text before and after the match, or display a Boolean value (True or False) that indicates whether a match is found.

`Select-String` can display all the text matches or stop after the first match in each input file. `Select-String` can be used to display all text that doesn't match the specified pattern.

You can also specify that `Select-String` should expect a particular character encoding, such as when you're searching files of Unicode text. `Select-String` uses the byte-order-mark (BOM) to detect the encoding format of the file. If the file has no BOM, it assumes the encoding is UTF8.

PARAMETERS

-AllMatches <System.Management.Automation.SwitchParameter>

Indicates that the cmdlet searches for more than one match in each line of text. Without this parameter, `Select-String` finds only the first match in each line of text.

When `Select-String` finds more than one match in a line of text, it still emits only one MatchInfo object for the line, but the Matches property of the object contains all the matches.

> [!NOTE] > This parameter is ignored when used in combination with the SimpleMatch parameter. If you wish > to return all matches and the pattern that you are searching for contains regular expression > characters, you must escape those characters rather than using SimpleMatch . See > about_Regular_Expressions (../Microsoft.PowerShell.Core/About/about_Regular_Expressions.md)for > more information about escaping regular expressions.

-CaseSensitive <System.Management.Automation.SwitchParameter>

Indicates that the cmdlet matches are case-sensitive. By default, matches aren't case-sensitive.

-Context <System.Int32[]>

Captures the specified number of lines before and after the line that matches the pattern.

If you enter one number as the value of this parameter, that number determines the number of lines captured before and after the match. If you enter two numbers as the value, the first number determines the number of lines before the match and the second number determines the number of lines after the match. For example, `-Context 2,3`.

In the default display, lines with a match are indicated by a right angle bracket (`>`) (ASCII 62) in the first column of the display. Unmarked lines are the context.

The Context parameter doesn't change the number of objects generated by `Select-String`. `Select-String` generates one MatchInfo

(/dotnet/api/microsoft.powershell.commands.matchinfo)object for each match. The context is stored as an array of strings in the Context property of the object.

When the output of a `Select-String` command is sent down the pipeline to another `Select-String` command, the receiving command searches only the text in the matched line. The matched line is the value of the Line property of the MatchInfo object, not the text in the context lines. As a result, the Context parameter isn't valid on the receiving `Select-String` command.

When the context includes a match, the MatchInfo object for each match includes all the context lines, but the overlapping lines appear only once in the display.

-Encoding <System.String>

Specifies the type of encoding for the target file. The default value is `default`.

The acceptable values for this parameter are as follows:

- `ascii` Uses ASCII (7-bit) character set.

- `bigendianunicode` Uses UTF-16 with the big-endian byte order.

- `default` Uses the encoding that corresponds to the system's active code page (usually ANSI).

- `oem` Uses the encoding that corresponds to the system's current OEM code page.

- `unicode` Uses UTF-16 with the little-endian byte order.

- `utf7` Uses UTF-7.

- `utf8` Uses UTF-8.

- `utf32` Uses UTF-32 with the little-endian byte order.

-Exclude <System.String[]>

Exclude the specified items. The value of this parameter qualifies the Path parameter. Enter a path element or pattern, such as `*.txt`. Wildcards are permitted.

-Include <System.String[]>

Includes the specified items. The value of this parameter qualifies the Path parameter. Enter a path element or pattern, such as `*.txt`. Wildcards are permitted.

-InputObject <System.Management.Automation.PSObject>

Specifies the text to be searched. Enter a variable that contains the text, or type a command or expression that gets the text.

Using the InputObject parameter isn't the same as sending strings down the pipeline to `Select-String`.

When you pipe more than one string to the `Select-String` cmdlet, it searches for the specified text in each string and returns each string that contains the search text.

When you use the InputObject parameter to submit a collection of strings, `Select-String` treats the collection as a single combined string. `Select-String` returns the strings as a unit if it finds the search text in any string.

-List <System.Management.Automation.SwitchParameter>

Only the first instance of matching text is returned from each input file.
This is the most efficient way to retrieve a list of files that have
contents matching the regular expression.

By default, `Select-String` returns a MatchInfo object for each match it
finds.

-LiteralPath <System.String[]>

Specifies the path to the files to be searched. The value of the
LiteralPath parameter is used exactly as it's typed. No characters are
interpreted as wildcards. If the path includes escape characters, enclose
it in single quotation marks. Single quotation marks tell PowerShell not
to interpret any characters as escape sequences. For more information, see
about_Quoting_Rules
(../Microsoft.Powershell.Core/About/about_Quoting_Rules.md).

-NotMatch <System.Management.Automation.SwitchParameter>

The NotMatch parameter finds text that doesn't match the specified pattern.

-Path <System.String[]>

Specifies the path to the files to search. Wildcards are permitted. The
default location is the local directory.

Specify files in the directory, such as `log1.txt`, ` .doc`, or ` .*`. If
you specify only a directory, the command fails.

-Pattern <System.String[]>

Specifies the text to find on each line. The pattern value is treated as a
regular expression.

To learn about regular expressions, see about_Regular_Expressions
(../Microsoft.PowerShell.Core/About/about_Regular_Expressions.md).

-Quiet <System.Management.Automation.SwitchParameter>

    Indicates that the cmdlet returns a Boolean value (True or False), instead
    of a MatchInfo object. The value is True if the pattern is found;
    otherwise the value is False.

-SimpleMatch <System.Management.Automation.SwitchParameter>

    Indicates that the cmdlet uses a simple match rather than a regular
    expression match. In a simple match, `Select-String` searches the input
    for the text in the Pattern parameter. It doesn't interpret the value of
    the Pattern parameter as a regular expression statement.

    Also, when SimpleMatch is used, the Matches property of the MatchInfo
    object returned is empty.

    > [!NOTE] > When this parameter is used with the AllMatches parameter, the
    AllMatches is ignored.

<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: Find a case-sensitive match ------------

```
'Hello', 'HELLO' | Select-String -Pattern 'HELLO' -CaseSensitive -SimpleMatch
```

The text strings Hello and HELLO are sent down the pipeline to the
`Select-String` cmdlet. `Select-String` uses the Pattern parameter to specify
HELLO . The CaseSensitive parameter specifies that the case must match only
the upper-case pattern. SimpleMatch is an optional parameter and specifies
that the string in the pattern isn't interpreted as a regular expression.
`Select-String` displays HELLO in the PowerShell console.

------------ Example 2: Find matches in text files ------------

```
Get-Alias | Out-File -FilePath .\Alias.txt
Get-Command | Out-File -FilePath .\Command.txt
Select-String -Path .\*.txt -Pattern 'Get-'
```

```
Alias.txt:8:Alias           cat -> Get-Content
Alias.txt:28:Alias           dir -> Get-ChildItem
Alias.txt:43:Alias           gal -> Get-Alias
Command.txt:966:Cmdlet       Get-Acl
Command.txt:967:Cmdlet       Get-Alias
```

In this example, `Get-Alias` and `Get-Command` are used with the `Out-File` cmdlet to create two text files in the current directory, Alias.txt and Command.txt .

`Select-String` uses the Path parameter with the asterisk (`*`) wildcard to search all files in the current directory with the file name extension `.txt`. The Pattern parameter specifies the text to match Get- . `Select-String` displays the output in the PowerShell console. The file name and line number precede each line of content that contains a match for the Pattern parameter.

--------------- Example 3: Find a pattern match ---------------

```
Select-String -Path "$PSHOME\en-US\*.txt" -Pattern '\?'
```

```
C:\Program Files\PowerShell\6\en-US\default.help.txt:27:    beginning at
https://go.microsoft.com/fwlink/?LinkID=108518.
C:\Program Files\PowerShell\6\en-US\default.help.txt:50:    or go to:
https://go.microsoft.com/fwlink/?LinkID=210614
```

The `Select-String` cmdlet uses two parameters, Path and Pattern . The Path parameter uses the variable `$PSHOME` that specifies the PowerShell directory. The remainder of the path includes the subdirectory en-US and specifies each

`*.txt` file in the directory. The Pattern parameter specifies to match a question mark (`?`) in each file. A backslash (``) is used as an escape character and is necessary because the question mark (`?`) is a regular expression quantifier. `Select-String` displays the output in the PowerShell console. The file name and line number precede each line of content that contains a match for the Pattern parameter.

---------- Example 4: Use Select-String in a function ----------

```
function Search-Help
{
   $PSHelp = "$PSHOME\en-US\*.txt"
   Select-String -Path $PSHelp -Pattern 'About_'
}

Search-Help

C:\Windows\System32\WindowsPowerShell\v1.0\en-US\about_ActivityCommonParameters
.help.txt:2:   about_ActivityCommonParameters
C:\Windows\System32\WindowsPowerShell\v1.0\en-US\about_ActivityCommonParameters
.help.txt:31:  see about_WorkflowCommonParameters.
C:\Windows\System32\WindowsPowerShell\v1.0\en-US\about_ActivityCommonParameters
.help.txt:33:  about_CommonParameters.
```

The function is created on the PowerShell command line. The `Function` command uses the name `Search-Help`. Press Enter to begin adding statements to the function. From the `>>` prompt, add each statement and press Enter as shown in the example. After the closing bracket is added, you're returned to a PowerShell prompt.

The function contains two commands. The `$PSHelp` variable stores the path to the PowerShell help files. `$PSHOME` is the PowerShell installation directory with the subdirectory en-US that specifies each `*.txt` file in the directory.

The `Select-String` command in the function uses the Path and Pattern parameters. The Path parameter uses the `$PSHelp` variable to get the path. The Pattern parameter uses the string About_ as the search criteria.

To run the function, type `Search-Help`. The function's `Select-String` command displays the output in the PowerShell console.

---- Example 5: Search for a string in a Windows event log ----

```
$Events = Get-WinEvent -LogName Application -MaxEvents 50
$Events | Select-String -InputObject {$_.message} -Pattern 'Failed'
```

The `Get-WinEvent` cmdlet uses the LogName parameter to specify the Application log. The MaxEvents parameter gets the 50 most recent events from the log. The log content is stored in the variable named `$Events`.

The `$Events` variable is sent down the pipeline to the `Select-String` cmdlet. `Select-String` uses the InputObject parameter. The `$_` variable represents the current object and `message` is a property of the event. The Pattern parameter species the string Failed and searches for matches in `$_.message`. `Select-String` displays the output in the PowerShell console.

---------- Example 6: Find a string in subdirectories ----------

```
Get-ChildItem -Path C:\Windows\System32\*.txt -Recurse | Select-String
-Pattern 'Microsoft' -CaseSensitive
```

`Get-ChildItem` uses the Path parameter to specify C:\Windows\System32*.txt . The Recurse parameter includes the subdirectories. The objects are sent down the pipeline to `Select-String`.

`Select-String` uses the Pattern parameter and specifies the string Microsoft . The CaseSensitive parameter is used to match the exact case of the string. `Select-String` displays the output in the PowerShell console.

> [!NOTE] > Dependent upon your permissions, you might see Access denied messages in the output.

----- Example 7: Find strings that do not match a pattern -----

```
Get-Command | Out-File -FilePath .\Command.txt
Select-String -Path .\Command.txt -Pattern 'Get', 'Set'  -NotMatch
```

The `Get-Command` cmdlet sends objects down the pipeline to the `Out-File` to create the Command.txt file in the current directory. `Select-String` uses the Path parameter to specify the Command.txt file. The Pattern parameter specifies Get and Set as the search pattern. The NotMatch parameter excludes Get and Set from the results. `Select-String` displays the output in the PowerShell console that doesn't include Get or Set .

-------- Example 8: Find lines before and after a match --------

```
Get-Command | Out-File -FilePath .\Command.txt
Select-String -Path .\Command.txt -Pattern 'Get-Computer' -Context 2, 3
```

```
Command.txt:1186:Cmdlet          Get-CmsMessage            3.0.0.0
Microsoft.PowerShell.Security
  Command.txt:1187:Cmdlet          Get-Command               3.0.0.0
Microsoft.PowerShell.Core
> Command.txt:1188:Cmdlet          Get-ComputerInfo          3.1.0.0
Microsoft.PowerShell.Management
> Command.txt:1189:Cmdlet          Get-ComputerRestorePoint  3.1.0.0
Microsoft.PowerShell.Management
  Command.txt:1190:Cmdlet          Get-Content               3.1.0.0
Microsoft.PowerShell.Management
  Command.txt:1191:Cmdlet          Get-ControlPanelItem      3.1.0.0
Microsoft.PowerShell.Management
  Command.txt:1192:Cmdlet          Get-Counter               3.0.0.0
Microsoft.PowerShell.Diagnostics
```

The `Get-Command` cmdlet sends objects down the pipeline to the `Out-File` to create the Command.txt file in the current directory. `Select-String` uses the Path parameter to specify the Command.txt file. The Pattern parameter specifies `Get-Computer` as the search pattern. The Context parameter uses two values, before and after, and marks pattern matches in the output with an angle bracket (`>`). The Context parameter outputs the two lines before the first pattern match and three lines after the last pattern match.

------------ Example 9: Find all pattern matches ------------

```
$A = Get-ChildItem -Path "$PSHOME\en-US\*.txt" | Select-String -Pattern
'PowerShell'
$A

C:\Windows\System32\WindowsPowerShell\v1.0\en-US\about_ActivityCommonParameters
.help.txt:5:    Describes the parameters that Windows PowerShell
C:\Windows\System32\WindowsPowerShell\v1.0\en-US\about_ActivityCommonParameters
.help.txt:9:    Windows PowerShell Workflow adds the activity common

$A.Matches

Groups   : {0}
Success  : True
Name     : 0
Captures : {0}
Index    : 4
Length   : 10
Value    : PowerShell

$A.Matches.Length

2073

$B = Get-ChildItem -Path "$PSHOME\en-US\*.txt" | Select-String -Pattern
```

```
'PowerShell' -AllMatches
$B.Matches.Length

2200
```

The `Get-ChildItem` cmdlet uses the Path parameter. The Path parameter uses the variable `$PSHOME` that specifies the PowerShell directory. The remainder of the path includes the subdirectory en-US and specifies each `*.txt` file in the directory. The `Get-ChildItem` objects are stored in the `$A` variable.
The `$A` variable is sent down the pipeline to the `Select-String` cmdlet.
`Select-String` uses the Pattern parameter to search each file for the string PowerShell .

From the PowerShell command line, the `$A` variable contents are displayed.
There's a line that contains two occurrences of the string PowerShell .

The `$A.Matches` property lists the first occurrence of the pattern PowerShell on each line.

The `$A.Matches.Length` property counts the first occurrence of the pattern PowerShell on each line.

The `$B` variable uses the same `Get-ChildItem` and `Select-String` cmdlets, but adds the AllMatches parameter. AllMatches finds each occurrence of the pattern PowerShell on each line. The objects stored in the `$A` and `$B` variables are identical.

The `$B.Matches.Length` property increases because for each line, every occurrence of the pattern PowerShell is counted.
Example 10 - Convert pipeline objects to strings using `Out-String`

```
PS> $hash = @{
   Name = 'foo'
```

```
        Category = 'bar'
    }

    # !! NO output, due to .ToString() conversion
    $hash | Select-String -Pattern 'foo'

    # Out-String converts the output to a single multi-line string object
    PS> $hash | Out-String | Select-String -Pattern 'foo'

    Name                           Value
    ----                           -----
    Name                           foo
    Category                       bar

    # Out-String -Stream converts the output to a multiple single-line string
    objects
    PS> $hash | Out-String -Stream | Select-String -Pattern 'foo'

    Name                           foo
```

Piping to `Out-String -Stream` converts the formatted output into a multiple single-line string objects. This means that when `Select-String` finds a match it outputs only the matching line.

REMARKS

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

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

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

For online help, type: "get-help Select-String -online"