PowerShell Get-Help on command 'Write-Progress'

PS C:\Users\wahid> Get-Help Write-Progress

NAME

Write-Progress

SYNOPSIS

Displays a progress bar within a PowerShell command window.

SYNTAX

Write-Progress [-Activity] <System.String> [[-Status] <System.String>] [[-Id] <System.Int32>] [-Completed] [-CurrentOperation <System.String>] [-ParentId <System.Int32>] [-PercentComplete <System.Int32>] [-SecondsRemaining <System.Int32>] [-SourceId <System.Int32>] [<CommonParameters>]

DESCRIPTION

The `Write-Progress` cmdlet displays a progress bar in a PowerShell command window that depicts the status of a running command or script. You can select the indicators that the bar reflects and the text that appears above and below the progress bar.

PARAMETERS

-Activity <System.String>

Specifies the first line of text in the heading above the status bar. This text describes the activity whose progress is being reported.

-Completed <System.Management.Automation.SwitchParameter>

Indicates whether the progress bar is visible. If this parameter is omitted, `Write-Progress` displays progress information.

-CurrentOperation <System.String>

Specifies the line of text below the progress bar. This text describes the operation that's currently taking place.

-Id <System.Int32>

Specifies an ID that distinguishes each progress bar from the others. Use this parameter when you are creating more than one progress bar in a single command. If the progress bars don't have different IDs, they're superimposed instead of being displayed in a series. Negative values aren't allowed.

-ParentId <System.Int32>

Specifies the parent activity of the current activity. Use the value `-1` if the current activity has no parent activity.

-PercentComplete <System.Int32>

Specifies the percentage of the activity that's completed. Use the value `-1` if the percentage complete is unknown or not applicable.

-SecondsRemaining <System.Int32>

Specifies the projected number of seconds remaining until the activity is completed. Use the value `-1` if the number of seconds remaining is unknown or not applicable.

-SourceId <System.Int32>

    Specifies the source of the record. You can use this in place of Id but
    can't be used with other parameters like ParentId .

-Status <System.String>

    Specifies the second line of text in the heading above the status bar.
    This text describes current state of the activity.

<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: Display the progress of a For loop --------

```
for ($i = 1; $i -le 100; $i++ ) {
    Write-Progress -Activity "Search in Progress" -Status "$i% Complete:"
-PercentComplete $i
    Start-Sleep -Milliseconds 250
}
```

This command displays the progress of a `for` loop that counts from 1 to 100.

The `Write-Progress` cmdlet includes a status bar heading `Activity`, a status
line, and the variable `$i` (the counter in the `for` loop), which indicates
the relative completeness of the task.

----- Example 2: Display the progress of nested For loops -----

```
for($I = 0; $I -lt 10; $I++ ) {
    $OuterLoopProgressParameters = @{
        Activity         = 'Updating'
        Status           = 'Progress->'
        PercentComplete  = $I * 10
        CurrentOperation = 'OuterLoop'
    }
    Write-Progress @OuterLoopProgressParameters
    for($j = 1; $j -lt 101; $j++ ) {
        $InnerLoopProgressParameters = @{
            ID               = 1
            Activity         = 'Updating'
            Status           = 'Progress'
            PercentComplete  = $j
            CurrentOperation = 'InnerLoop'
        }
        Write-Progress @InnerLoopProgressParameters
        Start-Sleep -Milliseconds 25
    }
}
```

```
Updating
Progress ->
 [oooooooooooooooooooooooooooooooooooooooooooooooooooooooooooooooooooooo]
OuterLoop
Updating
Progress
 [oooooooooooooooooo                                                    ]
InnerLoop
```

This example displays the progress of two nested For loops, each of which is represented by a progress bar.

The `Write-Progress` command for the second progress bar includes the Id parameter that distinguishes it from the first progress bar.

Without the Id parameter, the progress bars would be superimposed on each other instead of being displayed one below the other.

- Example 3: Display the progress while searching for a string -

```
# Use Get-EventLog to get the events in the System log and store them in the
$Events variable.
$Events = Get-EventLog -LogName system
# Pipe the events to the ForEach-Object cmdlet.
$Events | ForEach-Object -Begin {
    # In the Begin block, use Clear-Host to clear the screen.
    Clear-Host
    # Set the $i counter variable to zero.
    $i = 0
    # Set the $out variable to an empty string.
    $out = ""
} -Process {
    # In the Process script block search the message property of each incoming
object for "bios".
    if($_.message -like "*bios*")
    {
        # Append the matching message to the out variable.
        $out=$out + $_.Message
    }
    # Increment the $i counter variable which is used to create the progress
bar.
    $i = $i+1
    # Determine the completion percentage
    $Completed = ($i/$Events.count) * 100
    # Use Write-Progress to output a progress bar.
    # The Activity and Status parameters create the first and second lines of
the progress bar
    # heading, respectively.
    Write-Progress -Activity "Searching Events" -Status "Progress:"
-PercentComplete $Completed
} -End {
    # Display the matching messages using the out variable.
    $out
}
```

This command displays the progress of a command to find the string "bios" in the System event log.

The PercentComplete parameter value is calculated by dividing the number of events that have been processed `$i` by the total number of events retrieved `$Events.count` and then multiplying that result by 100.

Example 4: Display progress for each level of a nested process

```
foreach ( $i in 1..10 ) {
  Write-Progress -Id 0 "Step $i"
  foreach ( $j in 1..10 ) {
    Write-Progress -Id 1 -ParentId 0 "Step $i - Substep $j"
    foreach ( $k in 1..10 ) {
      Write-Progress -Id 2  -ParentId 1 "Step $i - Substep $j - iteration $k"
      Start-Sleep -Milliseconds 150
    }
  }
}
```

```
Step 1
     Processing
    Step 1 - Substep 2
         Processing
        Step 1 - Substep 2 - Iteration 3
             Processing
```

In this example you can use the ParentId parameter to have indented output to show parent-child relationships in the progress of each step.

REMARKS

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

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

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

For online help, type: "get-help Write-Progress -online"