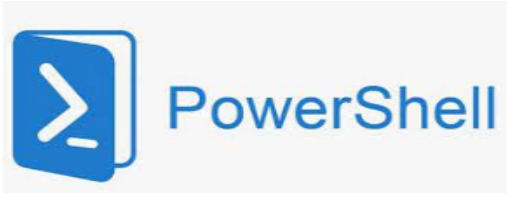

Full credit is given to the above companies including the OS that this PDF file was generated!

PowerShell Get-Help on command 'Register-ScheduledJob'

PS C:\Users\wahid> Get-Help Register-ScheduledJob

NAME

Register-ScheduledJob

SYNOPSIS

Creates a scheduled job.

SYNTAX

Register-ScheduledJob [-Name] <System.String> [-FilePath] <System.String>
[-ArgumentList <System.Object[]>] [-Authentication {Default | Basic |
Negotiate | NegotiateWithImplicitCredential | Credssp | Digest | Kerberos}]
[-Credential <System.Management.Automation.PSCredential>]
[-InitializationScript <System.Management.Automation.ScriptBlock>]
[-MaxResultCount <System.Int32>] [-RunAs32] [-RunEvery <System.TimeSpan>]
[-RunNow] [-ScheduledJobOption
<Microsoft.PowerShell.ScheduledJob.ScheduledJobOptions>] [-Trigger
<Microsoft.PowerShell.ScheduledJob.ScheduledJobTrigger[]>] [-Confirm]
[-WhatIf] [<CommonParameters>]

Register-ScheduledJob [-Name] <System.String> [-ScriptBlock]
<System.Management.Automation.ScriptBlock> [-ArgumentList <System.Object[]>]

[-Authentication {Default | Basic | Negotiate |
NegotiateWithImplicitCredential | Credssp | Digest | Kerberos}] [-Credential
<System.Management.Automation.PSCredential>] [-InitializationScript
<System.Management.Automation.ScriptBlock>] [-MaxResultCount <System.Int32>]
[-RunAs32] [-RunEvery <System.TimeSpan>] [-RunNow] [-ScheduledJobOption
<Microsoft.PowerShell.ScheduledJob.ScheduledJobOptions>] [-Trigger
<Microsoft.PowerShell.ScheduledJob.ScheduledJobTrigger[]>] [-Confirm]
[-WhatIf] [<CommonParameters>]

DESCRIPTION

The `Register-ScheduledJob` cmdlet creates scheduled jobs on the local computer.

A scheduled job is a Windows PowerShell background job that can be started automatically on a one-time or recurring schedule. Scheduled jobs are stored on disk and registered in Task Scheduler. The jobs can be managed in Task Scheduler or by using the Scheduled Job cmdlets in Windows PowerShell.

When a scheduled job starts, it creates an instance of the scheduled job. Scheduled job instances are identical to Windows PowerShell background jobs, except that the results are saved on disk. Use the Job cmdlets, such as `Start-Job`, `Get-Job`, and `Receive-Job` to start, view, and get the results of the job instances.

Use `Register-ScheduledJob` to create a new scheduled job. To specify the commands that the scheduled job runs, use the ScriptBlock parameter. To specify a script that the job runs, use the FilePath parameter.

Windows PowerShell-scheduled jobs use the same job triggers and job options that Task Scheduler uses for scheduled tasks.

The Trigger parameter of `Register-ScheduledJob` adds one or more job triggers

that start the job. The Trigger parameter is optional, so you can add triggers when you create the scheduled job, add job triggers later, add the RunNow parameter to start the job immediately, use the `Start-Job` cmdlet to start the job immediately at any time, or save the untriggered scheduled job as a template for other jobs.

The Options parameter lets you customize the options settings for the scheduled job. The Options parameter is optional, so you can set job options when you create the scheduled job or change them at any time. Because job option settings can prevent the scheduled job from running, review the job options and set them carefully.

`Register-ScheduledJob` is one of a collection of job scheduling cmdlets in the PSScheduledJob module that is included in Windows PowerShell.

For more information about Scheduled Jobs, see the About articles in the PSScheduledJob module. Import the PSScheduledJob module and then type: `Get-Help about_Scheduled*` or see about_Scheduled_Jobs (./about/about_scheduled_jobs.md).

This cmdlet was introduced in Windows PowerShell 3.0.

PARAMETERS

-ArgumentList <System.Object[]>

Specifies values for the parameters of the script that is specified by the FilePath parameter or for the command that is specified by the ScriptBlock parameter.

-Authentication <System.Management.Automation.Runspaces.AuthenticationMechanism>

Specifies the mechanism that is used to authenticate the user's credentials. The default value is Default.

The acceptable values for this parameter are:

- `Default`

- `Basic`

- `Credssp`

- `Digest`

- `Kerberos`

- `Negotiate`

- `NegotiateWithImplicitCredential`

For more information about the values of this parameter, see AuthenticationMechanism (/dotnet/api/system.management.automation.runspaces.authenticationmechanism).
> [!CAUTION] > Credential Security Service Provider (CredSSP) authentication, in which the user's credentials are > passed to a remote computer to be authenticated, is designed for commands that require > authentication on more than one resource, such as accessing a remote network share. This mechanism > increases the security risk of the remote operation. If the remote computer is compromised, the > credentials that are passed to it can be used to control the network session.

-Credential <System.Management.Automation.PSCredential>

Specifies a user account that has permission to run the scheduled job. The default is the current user.

Type a user name, such as User01 or Domain01\User01 , or enter a PSCredential object, such as one from the `Get-Credential` cmdlet. If you enter only a user name, you're prompted for a password.

Credentials are stored in a PSCredential (/dotnet/api/system.management.automation.pscredential)object and the password is stored as a SecureString (/dotnet/api/system.security.securestring).

> [!NOTE] > For more information about SecureString data protection, see > How secure is SecureString? (/dotnet/api/system.security.securestring#how-secure-is-securestring).

-FilePath <System.String>

Specifies a script that the scheduled job runs. Enter the path to a `.ps1` file on the local computer. To specify default values for the script parameters, use the ArgumentList parameter. Every `Register-ScheduledJob` command must use either the ScriptBlock or FilePath parameters.

-InitializationScript <System.Management.Automation.ScriptBlock>

Specifies the fully qualified path to a Windows PowerShell script (`.ps1`). The initialization script runs in the session that is created for the background job before the commands that are specified by the ScriptBlock parameter or the script that is specified by the FilePath parameter. You can use the initialization script to configure the session, such as adding files, functions, or aliases, creating directories, or checking for prerequisites.

To specify a script that runs the primary job commands, use the FilePath parameter.

If the initialization script generates an error, even a non-terminating

error, the current instance of the scheduled job doesn't run and its status is Failed .

-MaxResultCount <System.Int32>

Specifies how many job result entries are maintained for the scheduled job. The default value is 32.

Windows PowerShell saves the execution history and results of each triggered instance of the scheduled job on disk. The value of this parameter determines the number of job instance results that are saved for this scheduled job. When the number of job instance results exceeds this value, Windows PowerShell deletes the results of the oldest job instance to make room for the results of the newest job instance.

The job execution history and job results are saved in the `$HOME\AppData\Local\Microsoft\Windows\PowerShell\ScheduledJobs<JobName>\Output<Timestamp>` directories on the computer on which the job is created. To see the execution history, use the `Get-Job` cmdlet. To get the job results, use the `Receive-Job` cmdlet.

The MaxResultCount parameter sets the value of the ExecutionHistoryLength property of the scheduled job.

To delete the current execution history and job results, use the ClearExecutionHistory parameter of the `Set-ScheduledJob` cmdlet.

-Name <System.String>

Specifies a name for the scheduled job. The name is also used for all started instances of the scheduled job. The name must be unique on the computer. This parameter is required.

-RunAs32 <System.Management.Automation.SwitchParameter>

Runs the scheduled job in a 32-bit process.

-RunEvery <System.TimeSpan>

Used to specify how often to run the job. For example, use this option to run a job every 15 minutes.

-RunNow <System.Management.Automation.SwitchParameter>

Starts a job immediately, as soon as the `Register-ScheduledJob` cmdlet is run. This parameter eliminates the need to trigger Task Scheduler to run a Windows PowerShell script immediately after registration, and doesn't require users to create a trigger that specifies a starting date and time.

-ScheduledJobOption <Microsoft.PowerShell.ScheduledJob.ScheduledJobOptions>

Sets options for the scheduled job. Enter a ScheduledJobOptions object, such as one that you create by using the `New-ScheduledJobOption` cmdlet, or a hash table value.

You can set options for a scheduled job when you register the schedule job or use the `Set-ScheduledJobOption` or `Set-ScheduledJob` cmdlets to change the options.

Many of the options and their default values determine whether and when a scheduled job runs. Be sure to review these options before scheduling a job. For a description of the scheduled job options, including the default values, see `New-ScheduledJobOption`.

To submit a hash table, use the following keys. In the following hash table, the keys are shown with their default values.

`@{StartIfOnBattery=$False; StopIfGoingOnBattery=$True; WakeToRun=$False; StartIfNotIdle=$False; IdleDuration="00:10:00"; IdleTimeout="01:00:00"; StopIfGoingOffIdle=$True; RestartOnIdleResume=$False; ShowInTaskScheduler=$True; RunElevated=$False; RunWithoutNetwork=$False; DoNotAllowDemandStart=$False; MultipleInstancePolicy="IgnoreNew"}`

-ScriptBlock <System.Management.Automation.ScriptBlock>

Specifies the commands that the scheduled job runs. Enclose the commands in curly braces (`{}`) to create a script block. To specify default values for command parameters, use the ArgumentList parameter.

Every `Register-ScheduledJob` command must use either the ScriptBlock or FilePath parameters.

-Trigger <Microsoft.PowerShell.ScheduledJob.ScheduledJobTrigger[]>

Specifies the triggers for the scheduled job. Enter one or more ScheduledJobTrigger objects, such as the objects that the `New-JobTrigger` cmdlet returns, or a hash table of job trigger keys and values.

A job trigger starts the schedule job. The trigger can specify a one-time or recurring scheduled or an event, such as when a user logs on or Windows starts.

The Trigger parameter is optional. You can add a trigger when you create the scheduled job, use the `Add-JobTrigger`, `Set-JobTrigger`, or `Set-ScheduledJob` cmdlets to add or change job triggers later, or use the `Start-Job` cmdlet to start the scheduled job immediately. You can also create and maintain a scheduled job without a trigger that is used as a template.

To submit a hash table, use the following keys:

- Frequency : Daily, Weekly, AtStartup, AtLogon - At : Any valid time string - DaysOfWeek - Any combination of day names - Interval - Any valid frequency interval - RandomDelay : Any valid timespan string - User : Any valid user. Used only with the AtLogon frequency value

For example:

`@{Frequency="Once"; At="3am"; DaysOfWeek="Monday", "Wednesday";
Interval=2; RandomDelay="30minutes"; User="Domain1\User01"}`

-Confirm <System.Management.Automation.SwitchParameter>
Prompts you for confirmation before running the cmdlet.

-WhatIf <System.Management.Automation.SwitchParameter>
Shows what would happen if the cmdlet runs. The cmdlet isn't run.

<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: Create a scheduled job ----------

```
Register-ScheduledJob -Name "Archive-Scripts" -ScriptBlock {
  Get-ChildItem $HOME\*.ps1 -Recurse |
    Copy-Item -Destination "\\Server\Share\PSScriptArchive"
}
```

`Register-ScheduledJob` uses the Name parameter to create the
`Archive-Scripts` scheduled job. The ScriptBlock parameter runs
`Get-ChildItem` that searches the `$HOME` directory recursively for `.ps1`
files. The `Copy-Item` cmdlet copies the files to a directory specified by the
Destination parameter.

Because the scheduled job doesn't contain a trigger, it's not started
automatically. You can add job triggers with `Add-JobTrigger`, use the
`Start-Job` cmdlet to start the job on demand, or use the scheduled job as a
template for other scheduled jobs.

Example 2: Create a scheduled job with triggers and custom options

```
$O = New-ScheduledJobOption -WakeToRun -StartIfIdle -MultipleInstancePolicy
Queue
$T = New-JobTrigger -Weekly -At "9:00 PM" -DaysOfWeek Monday -WeeksInterval 2
$path = "\\Srv01\Scripts\UpdateVersion.ps1"
Register-ScheduledJob -Name "UpdateVersion" -FilePath $path
-ScheduledJobOption $O -Trigger $T
```

The `$O` variable stores the job option object that the `New-ScheduledJobOption` cmdlet created. The options start the scheduled job even if the computer isn't idle, wakes the computer to run the job, if necessary, and allows multiple instances of the job to run in a series.

The `$T` variable stores the result from the `New-JobTrigger` cmdlet to create job trigger that starts a job every other Monday at 9:00 PM.

The `$path` variable stores the path to the `UpdateVersion.ps1` script file.

`Register-ScheduledJob` uses the Name parameter to create the UpdateVersion scheduled job. The FilePath parameter uses `$path` to specify the script that the job runs. The ScheduledJobOption parameter uses the job options stored in `$O`. The Trigger parameter uses the job triggers stored in `$T`.

Example 3: Use hash tables to specify a trigger and scheduled job options

```
$T = @{
  Frequency="Weekly"
  At="9:00PM"
  DaysOfWeek="Monday"
  Interval=2
}
$O = @{
  WakeToRun=$true
```

```
  StartIfNotIdle=$false
  MultipleInstancePolicy="Queue"
}
Register-ScheduledJob -Trigger $T -ScheduledJobOption $O -Name UpdateVersion
-FilePath "\\Srv01\Scripts\Update-Version.ps1"
```

----- Example 4: Create scheduled jobs on remote computers -----

```
$Cred = Get-Credential
$O = New-ScheduledJobOption -WakeToRun -StartIfIdle -MultipleInstancePolicy
Queue
$T = New-JobTrigger -Weekly -At "9:00 PM" -DaysOfWeek Monday -WeeksInterval 2
Invoke-Command -ComputerName (Get-Content Servers.txt) -Credential $Cred
-ScriptBlock {
  $params = @{
      Name = "Get-EnergyData"
      FilePath = "\\Srv01\Scripts\Get-EnergyData.ps1"
      ScheduledJobOption = $using:O
      Trigger = $using:T
  }
  Register-ScheduledJob @params
}
```

The `$Cred` variable stores credentials in a PSCredential object for a user with permissions to create scheduled jobs. The `$O` variable stores the job options created with `New-ScheduledJobOption`. The `$T` variable stores the job triggers created with `New-JobTrigger`.

The `Invoke-Command` cmdlet uses the ComputerName parameter to get a list of server names from the `Servers.txt` file. The Credential parameter gets the credential object stored in `$Cred`. The ScriptBlock parameter runs a `Register-ScheduledJob` command on the computers in the `Servers.txt` file.

The parameters for `Register-ScheduledJob` are defined by `$params`. The Name parameters specifies the job is named `Get-EnergyData` on each remote computer. FilePath provides the location of the `EnergyData.ps1` script. The script is located on a file server that is available to all participating computers.The job runs on the schedule specified by the job triggers in `$T` and the job options in `$O`.

The `Register-ScheduledJob @params` command creates the scheduled job with the parameters from the script block.

Example 5: Create a scheduled job that runs a script on remote computers

```
$Admin = Get-Credential
$T = New-JobTrigger -Weekly -At "9:00 PM" -DaysOfWeek Monday -WeeksInterval 2
Register-ScheduledJob -Name "CollectEnergyData" -Trigger $T -MaxResultCount 99
-ScriptBlock {
  $params = @{
    AsJob = $true
    ComputerName = (Get-Content Servers.txt)
    FilePath = '\\Srv01\Scripts\Get-EnergyData.ps1'
    Credential = $using:Admin
    Authentication = 'CredSSP'
  }
  Invoke-Command @params
}
```

The `$Admin` variable stores credentials for a user with permissions to run the commands in a PSCredential object. The `$T` variable stores the job triggers created with `New-JobTrigger`.

The `Register-ScheduledJob` cmdlet uses the Name parameter to create the CollectEnergyData scheduled job on the local computer. The Trigger parameter specifies the job triggers in `$T` and the MaxResultCount parameter increases

the number of saved results to 99.

The ScriptBlock parameter defines the `Invoke-Command` parameters with `$params`. The AsJob parameter creates the background job object on the local computer, even though the `Energydata.ps1` script runs on the remote computers. The ComputerName parameter gets a list of server names from the `Servers.txt` file. The Credential parameter specifies a user account that has permission to run scripts on the remote computers. And, the Authentication parameter specifies a value of CredSSP to allow delegated credentials.

The `Invoke-Command @params` runs the command with the parameters from the script block.

REMARKS

To see the examples, type: "get-help Register-ScheduledJob -examples".
For more information, type: "get-help Register-ScheduledJob -detailed".
For technical information, type: "get-help Register-ScheduledJob -full".
For online help, type: "get-help Register-ScheduledJob -online"