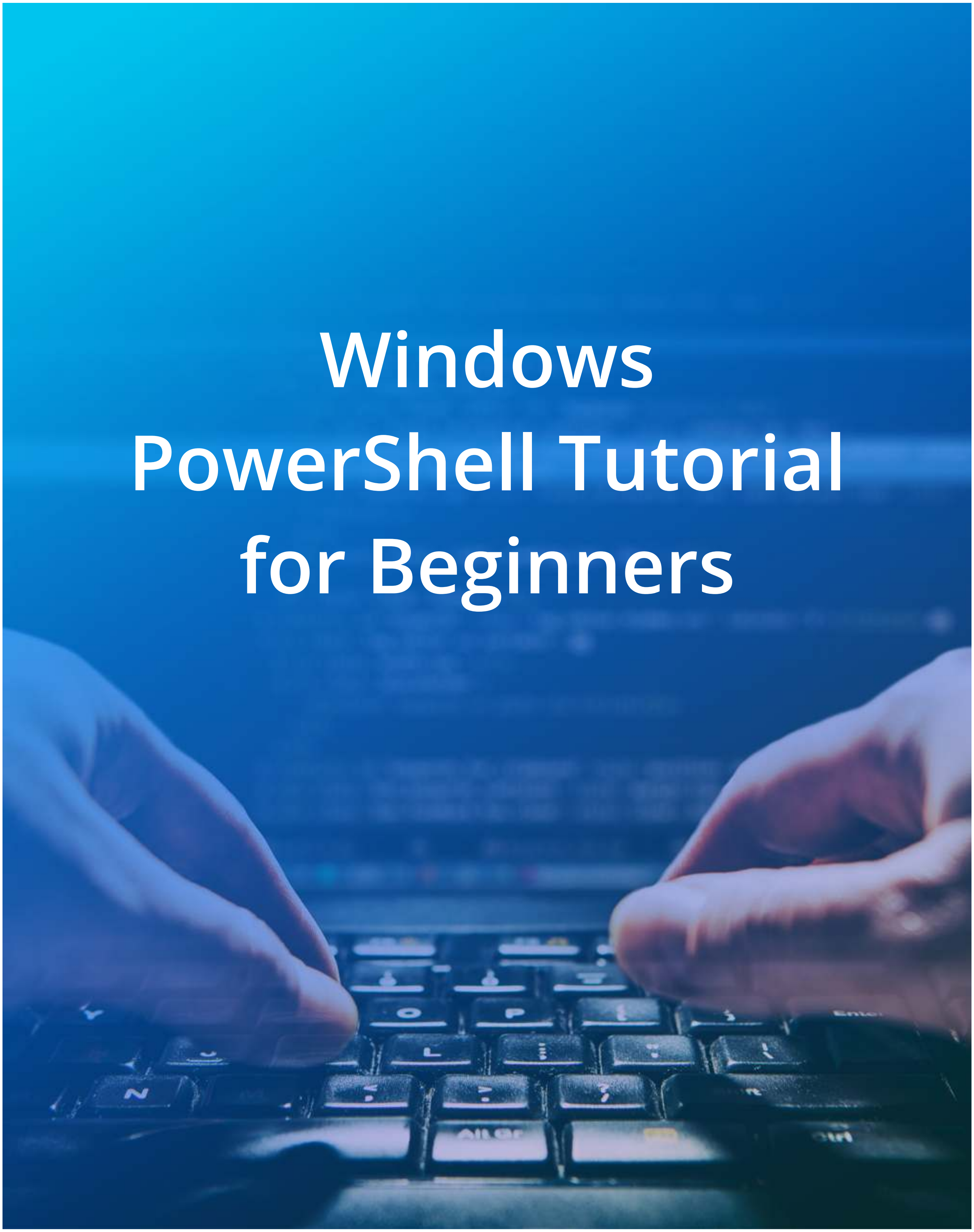
netwrix



Windows PowerShell Tutorial for Beginners

Table of Contents

Introduction

Automate it. Now, where's that script...
Warning: PowerShell is addictive.

Windows PowerShell is an object-oriented automation engine and scripting language with an interactive command-line shell designed to help IT professionals configure systems and automate administrative tasks. You can find it in every modern Windows OS starting with Windows 2008R2.

Learning Windows PowerShell is like learning to use a universal multi-tool. In this eBook, we'll walk you through PowerShell scripting basics, show you PowerShell commands and scripts for performing the most common administrative tasks, and explain how you can schedule your PowerShell scripts and commands.

So, let's start learning PowerShell!

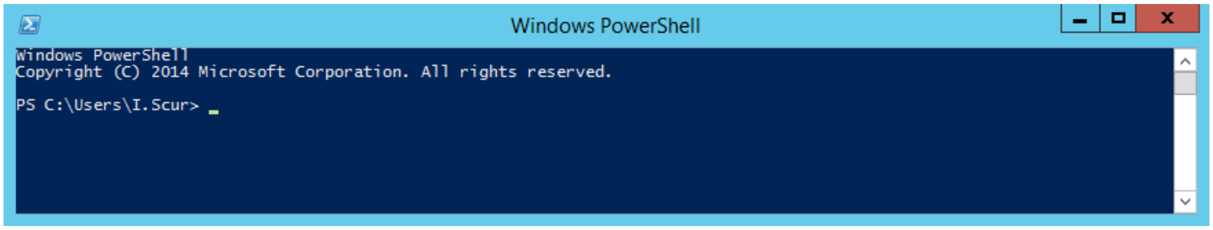
1. PowerShell Scripting Basics

In this part, we'll cover PowerShell scripting basics so you can more easily perform virtually any administration task in your Windows IT environment.

1.1 Launching the PowerShell

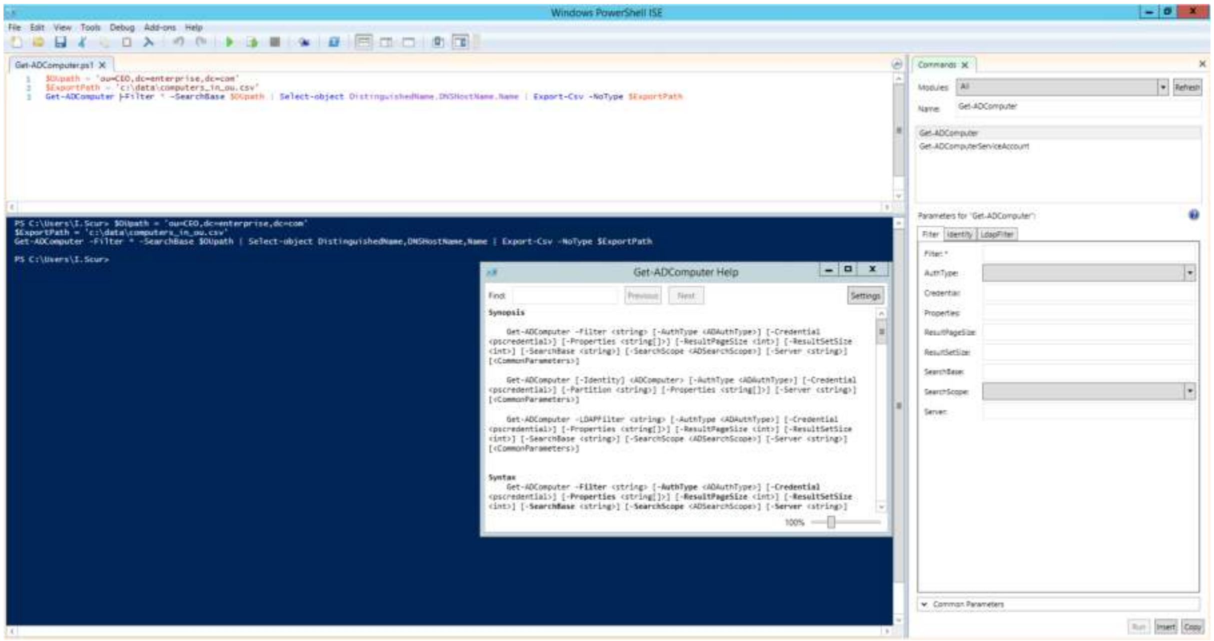
PowerShell offers both a command-line option and an integrated scripting environment (ISE):

- To launch the PowerShell command line, type **powershell.exe** in the Windows Start menu. You'll see a screen like the following:



```
Windows PowerShell
Copyright (C) 2014 Microsoft Corporation. All rights reserved.

PS C:\Users\I.Scur> _
```

- To launch the PowerShell ISE, type **powershell_ise.exe** in the Start menu. Using the PowerShell ISE is the preferred way to work with the scripting language because it provides syntax highlighting, auto-filling of commands and other automation features that simplify script development and testing.



```
$OUpath = 'ou=CEO,dc=enterprise,dc=com'
$ExportPath = 'c:\data\computers_in_ou.csv'
Get-ADComputer -Filter * -SearchBase $OUpath | Select-object DistinguishedName,DNSHostName,Name | Export-Csv -NoType $ExportPath
```

```
PS C:\Users\I.Scur> $OUpath = 'ou=CEO,dc=enterprise,dc=com'
$ExportPath = 'c:\data\computers_in_ou.csv'
Get-ADComputer -Filter * -SearchBase $OUpath | Select-object DistinguishedName,DNSHostName,Name | Export-Csv -NoType $ExportPath

PS C:\Users\I.Scur>
```

```
Synopsis

    Get-ADComputer -Filter <string> [-AuthType <ADAuthType>] [-Credential
<pscredential>] [-Properties <string[]>] [-ResultPageSize <int>] [-ResultSetSize
<int>] [-SearchBase <string>] [-SearchScope <ADSearchScope>] [-Server <string>]
[<CommonParameters>]

    Get-ADComputer [-Identity] <ADComputer> [-AuthType <ADAuthType>] [-Credential
<pscredential>] [-Partition <string>] [-Properties <string[]>] [-Server <string>]
[<CommonParameters>]

    Get-ADComputer -LDAPFilter <string> [-AuthType <ADAuthType>] [-Credential
<pscredential>] [-Properties <string[]>] [-ResultPageSize <int>] [-ResultSetSize
<int>] [-SearchBase <string>] [-SearchScope <ADSearchScope>] [-Server <string>]
[<CommonParameters>]


Syntax
    Get-ADComputer -Filter <string> [-AuthType <ADAuthType>] [-Credential
<pscredential>] [-Properties <string[]>] [-ResultPageSize <int>] [-ResultSetSize
<int>] [-SearchBase <string>] [-SearchScope <ADSearchScope>] [-Server <string>]
```

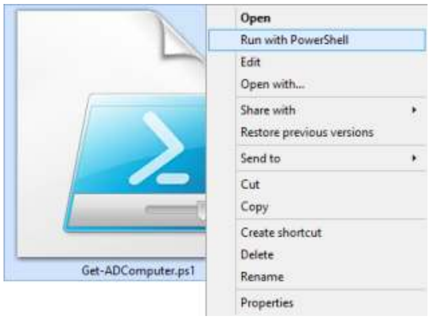
Modules: All
Name: Get-ADComputer

Get-ADComputer
Get-ADComputerServiceAccount

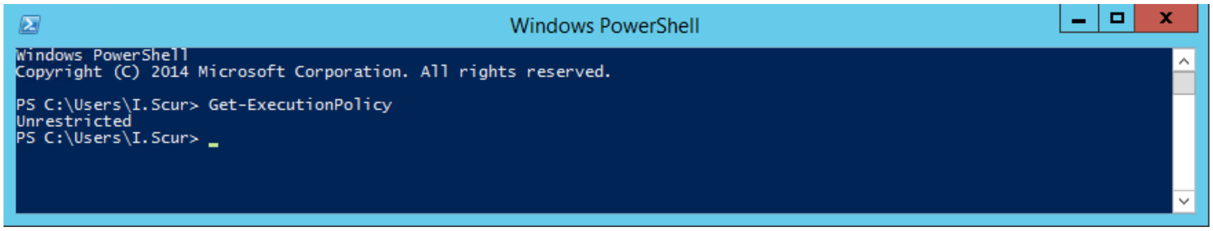
Filter: *
AuthType:
Credential:
Properties:
ResultPageSize:
ResultSetSize:
SearchBase:
SearchScope:
Server:

1.2 Preparing to Run PowerShell Scripts

PowerShell scripts are stored in.ps1 files. You cannot run a script by simply double-clicking a file; this design helps avoid accidental harm to your systems. Instead, to execute a script, right-click it and choose **Run with PowerShell**:



In addition, there is a policy that restricts script execution. You can check this policy by running the **Get-ExecutionPolicy** command in PowerShell:

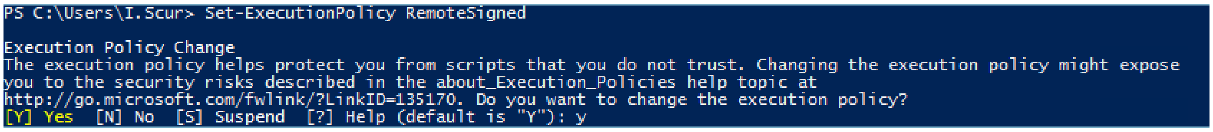


```
Windows PowerShell
Copyright (C) 2014 Microsoft Corporation. All rights reserved.

PS C:\Users\I.Scur> Get-ExecutionPolicy
Unrestricted
PS C:\Users\I.Scur> _
```

You will get one of the following values:

- **Restricted** — No scripts are allowed. This is the default setting, so you will see it the first time you run the command.
- **AllSigned** — You can run scripts signed by a trusted developer. Before executing, a script will ask you to confirm that you want to run it.
- **RemoteSigned** — You can run your own scripts or scripts signed by a trusted developer.
- **Unrestricted** — You can run any script you want.

To start working with PowerShell, you'll need to change the policy setting from Restricted to RemoteSigned using the **Set-ExecutionPolicy RemoteSigned** command:



```
PS C:\Users\I.Scur> Set-ExecutionPolicy RemoteSigned

Execution Policy Change
The execution policy helps protect you from scripts that you do not trust. Changing the execution policy might expose
you to the security risks described in the about_Execution_Policies help topic at
http://go.microsoft.com/fwlink/?LinkID=135170. Do you want to change the execution policy?
[Y] Yes  [N] No  [S] Suspend  [?] Help (default is "Y"): y
```

1.3 PowerShell Cmdlets

A cmdlet is a PowerShell command with a predefined function, similar to an operator in a programming language. Here are some key things to know about cmdlets:

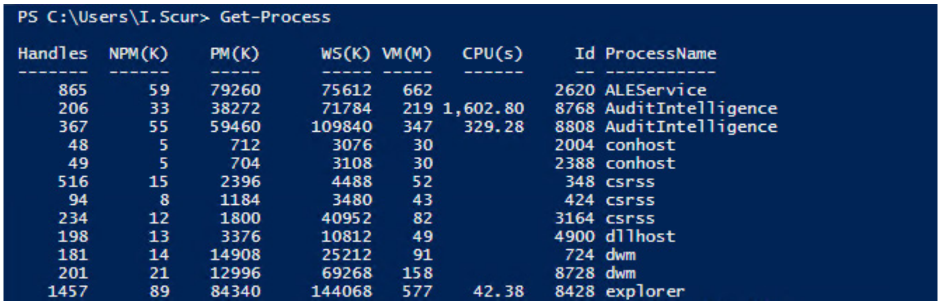
- There are system, user and custom cmdlets.
- Cmdlets output results as an object or as an array of objects.
- Cmdlets can get data for analysis or transfer data to another cmdlet using pipes (I'll discuss pipes more in a moment).
- Cmdlets are case-insensitive. For example, it doesn't matter whether you type **Get-ADUser**, **get-aduser** or **gEt-AdUsEr**.
- If you want to use several cmdlets in one string, you must separate them with a semicolon (;).

A cmdlet always consists of a verb (or a word that functions as a verb) and a noun, separated with a hyphen (the "verb-noun" rule). For example, some of the verbs include:

- **Get** — To get something
- **Set** — To define something
- **Start** — To run something
- **Stop** — To stop something that is running
- **Out** — To output something
- **New** — To create something ("new" is not a verb, of course, but it functions as one)

For practice, try executing the following cmdlets:

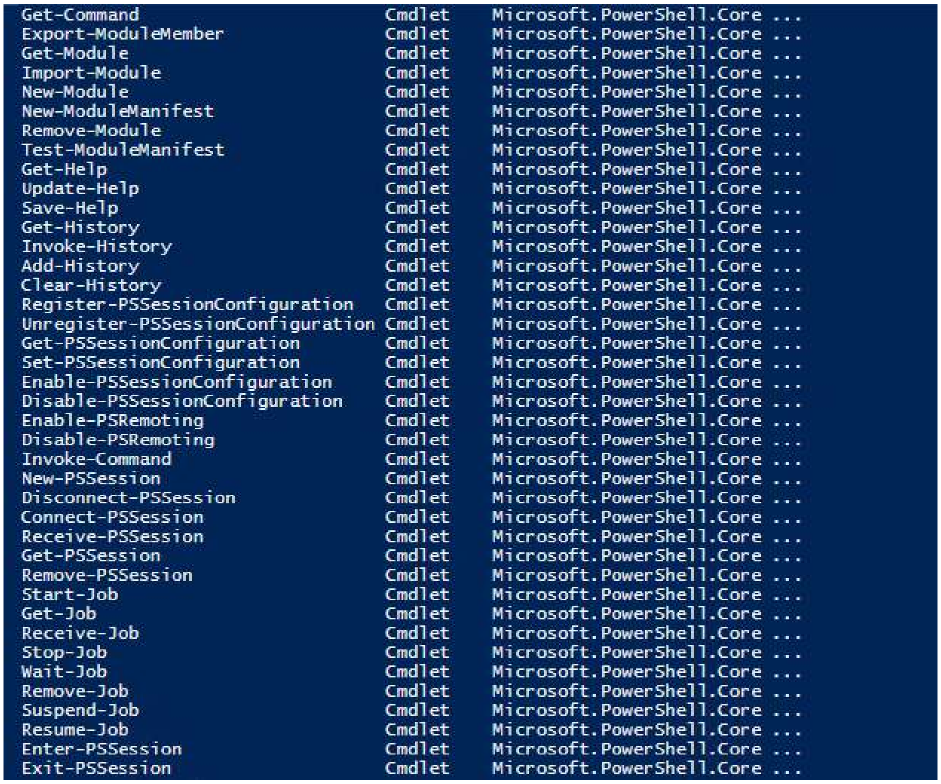
- **Get-Process** — Shows the processes currently running on your computer:



```
PS C:\Users\I.Scur> Get-Process

Handles  NPM(K)    PM(K)      WS(K) VM(M)   CPU(s)     Id ProcessName
-------  ------    -----      ----- -----   ------     -- -----------
    865      59    79260      75612   662            2620 ALEService
    206      33    38272      71784   219 1,602.80   8768 AuditIntelligence
    367      55    59460     109840   347   329.28   8808 AuditIntelligence
     48       5      712       3076    30            2004 conhost
     49       5      704       3108    30            2388 conhost
    516      15     2396       4488    52             348 csrss
     94       8     1184       3480    43             424 csrss
    234      12     1800      40952    82            3164 csrss
    198      13     3376      10812    49            4900 dllhost
    181      14    14908      25212    91             724 dwm
    201      21    12996      69268   158            8728 dwm
   1457      89    84340     144068   577    42.38   8428 explorer
```

- **Get-Service** — Shows the list of services with their status
- **Get-Content** — Shows the content of the file you specify (for example, **Get-Content C:\Windows\System32\drivers\etc\hosts**)

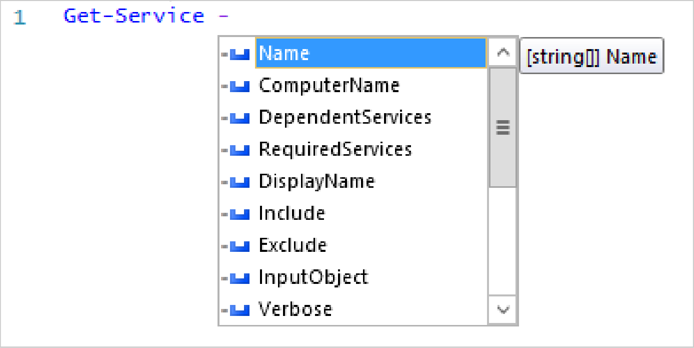
Good news — you don't need to memorize all cmdlets. You can list all cmdlets by executing the **Get Help -Category** cmdlet, which will return the following:



```
Get-Command                       Cmdlet    Microsoft.PowerShell.Core ...
Export-ModuleMember               Cmdlet    Microsoft.PowerShell.Core ...
Get-Module                        Cmdlet    Microsoft.PowerShell.Core ...
Import-Module                     Cmdlet    Microsoft.PowerShell.Core ...
New-Module                        Cmdlet    Microsoft.PowerShell.Core ...
New-ModuleManifest                Cmdlet    Microsoft.PowerShell.Core ...
Remove-Module                     Cmdlet    Microsoft.PowerShell.Core ...
Test-ModuleManifest               Cmdlet    Microsoft.PowerShell.Core ...
Get-Help                          Cmdlet    Microsoft.PowerShell.Core ...
Update-Help                       Cmdlet    Microsoft.PowerShell.Core ...
Save-Help                         Cmdlet    Microsoft.PowerShell.Core ...
Get-History                       Cmdlet    Microsoft.PowerShell.Core ...
Invoke-History                    Cmdlet    Microsoft.PowerShell.Core ...
Add-History                       Cmdlet    Microsoft.PowerShell.Core ...
Clear-History                     Cmdlet    Microsoft.PowerShell.Core ...
Register-PSSessionConfiguration   Cmdlet    Microsoft.PowerShell.Core ...
Unregister-PSSessionConfiguration Cmdlet    Microsoft.PowerShell.Core ...
Get-PSSessionConfiguration        Cmdlet    Microsoft.PowerShell.Core ...
Set-PSSessionConfiguration        Cmdlet    Microsoft.PowerShell.Core ...
Enable-PSSessionConfiguration     Cmdlet    Microsoft.PowerShell.Core ...
Disable-PSSessionConfiguration    Cmdlet    Microsoft.PowerShell.Core ...
Enable-PSRemoting                 Cmdlet    Microsoft.PowerShell.Core ...
Disable-PSRemoting                Cmdlet    Microsoft.PowerShell.Core ...
Invoke-Command                    Cmdlet    Microsoft.PowerShell.Core ...
New-PSSession                     Cmdlet    Microsoft.PowerShell.Core ...
Disconnect-PSSession              Cmdlet    Microsoft.PowerShell.Core ...
Connect-PSSession                 Cmdlet    Microsoft.PowerShell.Core ...
Receive-PSSession                 Cmdlet    Microsoft.PowerShell.Core ...
Get-PSSession                     Cmdlet    Microsoft.PowerShell.Core ...
Remove-PSSession                  Cmdlet    Microsoft.PowerShell.Core ...
Start-Job                         Cmdlet    Microsoft.PowerShell.Core ...
Get-Job                           Cmdlet    Microsoft.PowerShell.Core ...
Receive-Job                       Cmdlet    Microsoft.PowerShell.Core ...
Stop-Job                          Cmdlet    Microsoft.PowerShell.Core ...
Wait-Job                          Cmdlet    Microsoft.PowerShell.Core ...
Remove-Job                        Cmdlet    Microsoft.PowerShell.Core ...
Suspend-Job                       Cmdlet    Microsoft.PowerShell.Core ...
Resume-Job                        Cmdlet    Microsoft.PowerShell.Core ...
Enter-PSSession                   Cmdlet    Microsoft.PowerShell.Core ...
Exit-PSSession                    Cmdlet    Microsoft.PowerShell.Core ...
```

You can also create your own custom cmdlets.

Each cmdlet has several parameters that customize what it does. The PowerShell ISE will automatically suggest all valid parameters and their types after you type a cmdlet and a hyphen (-):

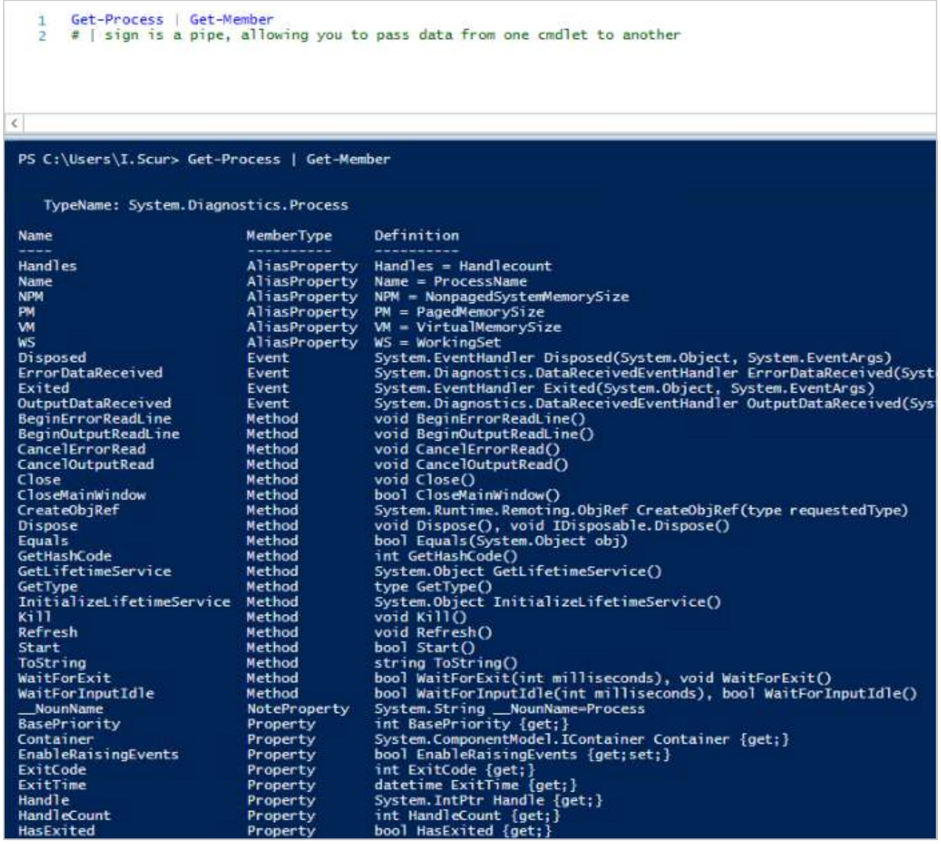


```
Get-Service -
```

For example, the following cmdlet shows all services whose names start with “W”:
Get-Service -Name W*

If you forget a cmdlet’s parameters, just use a script like the following, which will display the parameters for the **Get-Process** cmdlet:

```
Get-Process  |  Get-Member
```

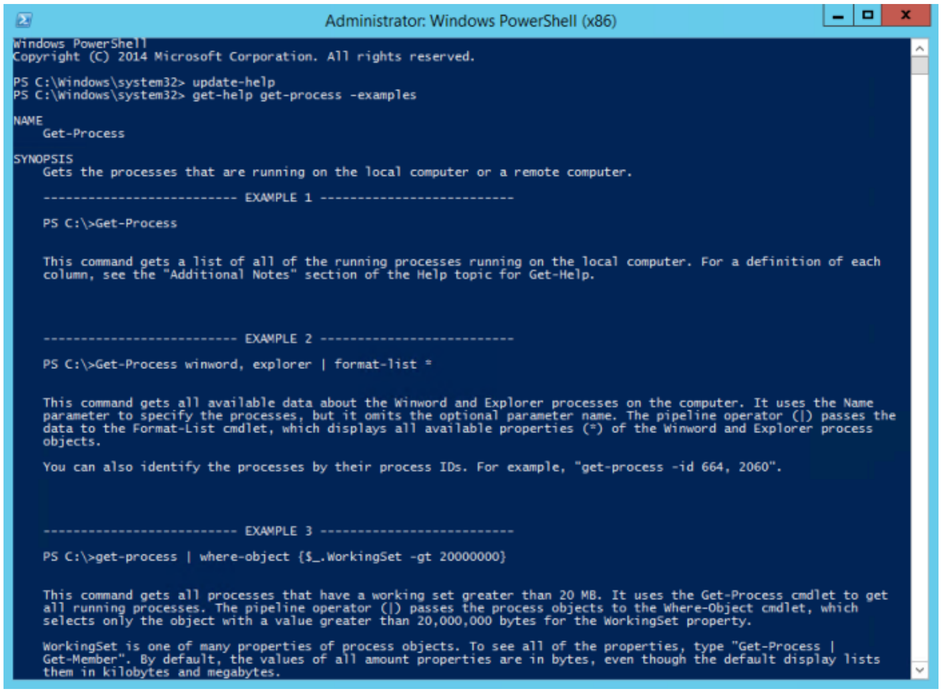


```
Get-Process | Get-Member
# | sign is a pipe, allowing you to pass data from one cmdlet to another
```

```
PS C:\Users\I.Scur> Get-Process | Get-Member


   TypeName: System.Diagnostics.Process

Name                       MemberType     Definition
----                       ----------     ----------
Handles                    AliasProperty  Handles = Handlecount
Name                       AliasProperty  Name = ProcessName
NPM                        AliasProperty  NPM = NonpagedSystemMemorySize
PM                         AliasProperty  PM = PagedMemorySize
VM                         AliasProperty  VM = VirtualMemorySize
WS                         AliasProperty  WS = WorkingSet
Disposed                   Event          System.EventHandler Disposed(System.Object, System.EventArgs)
ErrorDataReceived          Event          System.Diagnostics.DataReceivedEventHandler ErrorDataReceived(Syst
Exited                     Event          System.EventHandler Exited(System.Object, System.EventArgs)
OutputDataReceived         Event          System.Diagnostics.DataReceivedEventHandler OutputDataReceived(Sys
BeginErrorReadLine         Method         void BeginErrorReadLine()
BeginOutputReadLine        Method         void BeginOutputReadLine()
CancelErrorRead            Method         void CancelErrorRead()
CancelOutputRead           Method         void CancelOutputRead()
Close                      Method         void Close()
CloseMainWindow            Method         bool CloseMainWindow()
CreateObjRef               Method         System.Runtime.Remoting.ObjRef CreateObjRef(type requestedType)
Dispose                    Method         void Dispose(), void IDisposable.Dispose()
Equals                     Method         bool Equals(System.Object obj)
GetHashCode                Method         int GetHashCode()
GetLifetimeService         Method         System.Object GetLifetimeService()
GetType                    Method         type GetType()
InitializeLifetimeService  Method         System.Object InitializeLifetimeService()
Kill                       Method         void Kill()
Refresh                    Method         void Refresh()
Start                      Method         bool Start()
ToString                   Method         string ToString()
WaitForExit                Method         bool WaitForExit(int milliseconds), void WaitForExit()
WaitForInputIdle           Method         bool WaitForInputIdle(int milliseconds), bool WaitForInputIdle()
__NounName                 NoteProperty   System.String __NounName=Process
BasePriority               Property       int BasePriority {get;}
Container                  Property       System.ComponentModel.IContainer Container {get;}
EnableRaisingEvents        Property       bool EnableRaisingEvents {get;set;}
ExitCode                   Property       int ExitCode {get;}
ExitTime                   Property       datetime ExitTime {get;}
Handle                     Property       System.IntPtr Handle {get;}
HandleCount                Property       int HandleCount {get;}
HasExited                  Property       bool HasExited {get;}
```

If you still don’t find the cmdlet you need, you can make sure the help is current and then get examples for a cmdlet (such as **Get-Process**) using a script like this:

```
Update-Help  #to update the help data
Get-Help  Get-Process  -Examples
```



```
Windows PowerShell
Copyright (C) 2014 Microsoft Corporation. All rights reserved.

PS C:\Windows\system32> update-help
PS C:\Windows\system32> get-help get-process -examples

NAME
    Get-Process

SYNOPSIS
    Gets the processes that are running on the local computer or a remote computer.

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

    PS C:\>Get-Process


    This command gets a list of all of the running processes running on the local computer. For a definition of each
    column, see the "Additional Notes" section of the Help topic for Get-Help.




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

    PS C:\>Get-Process winword, explorer | format-list *


    This command gets all available data about the Winword and Explorer processes on the computer. It uses the Name
    parameter to specify the processes, but it omits the optional parameter name. The pipeline operator (|) passes the
    data to the Format-List cmdlet, which displays all available properties (*) of the Winword and Explorer process
    objects.

    You can also identify the processes by their process IDs. For example, "get-process -id 664, 2060".




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

    PS C:\>get-process | where-object {$_.WorkingSet -gt 20000000}


    This command gets all processes that have a working set greater than 20 MB. It uses the Get-Process cmdlet to get
    all running processes. The pipeline operator (|) passes the process objects to the Where-Object cmdlet, which
    selects only the object with a value greater than 20,000,000 bytes for the WorkingSet property.

    WorkingSet is one of many properties of process objects. To see all of the properties, type "Get-Process |
    Get-Member". By default, the values of all amount properties are in bytes, even though the default display lists
    them in kilobytes and megabytes.
```

You can also use aliases, which are shortened cmdlet names. For instance, instead of **Get-Help** you can use just **Help**. Try running the following two commands and see whether you get the same result:

```
Start-Process  notepad
```

```
start  notepad
```

Similarly, to stop this process, you can use either of the following commands:

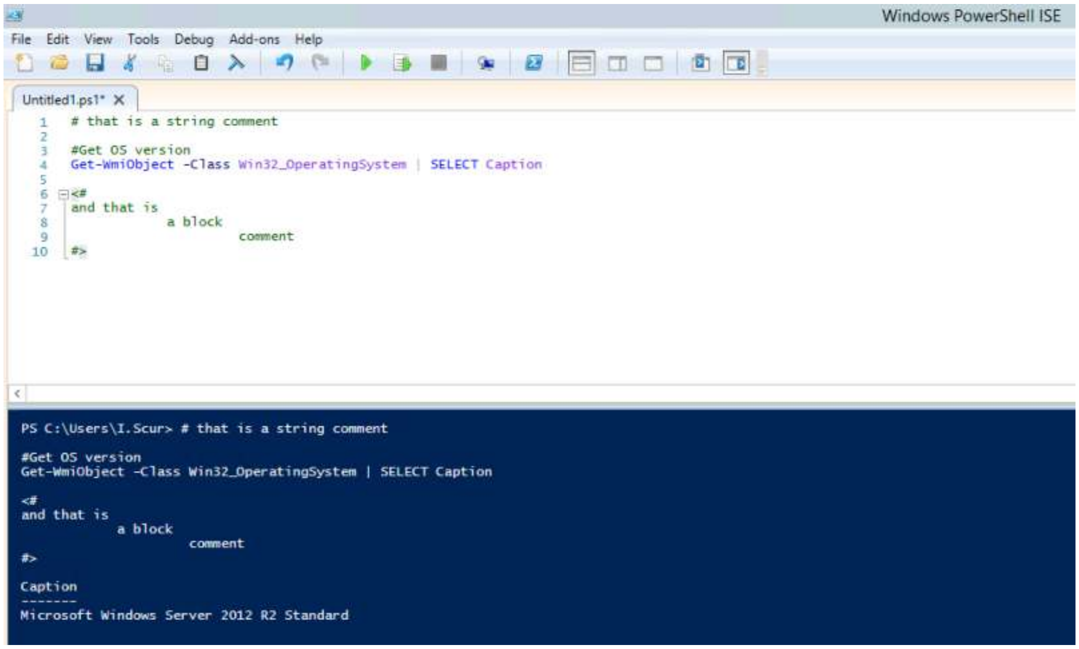
```
Stop-Process  -Name  notepad
```

```
spps  -Name  notepad
```

To see all aliases, execute the **Get-Alias** cmdlet.

1.4 Comments

Leaving comments in a script will help you — and your colleagues — better understand what the script does. A string comment is a single line that starts with a number sign (#); block comments spread across multiple lines, starting and ending with number signs and angle brackets:



```
# that is a string comment

#Get OS version
Get-WmiObject -Class Win32_OperatingSystem | SELECT Caption

<#
and that is
            a block
                     comment
#>
```

```
PS C:\Users\I.Scur> # that is a string comment

#Get OS version
Get-WmiObject -Class Win32_OperatingSystem | SELECT Caption

<#
and that is
            a block
                     comment
#>

Caption
-------
Microsoft Windows Server 2012 R2 Standard
```

1.5 Pipes

A pipe passes data from one cmdlet to another. I used a pipe earlier to get all properties of an object.

For example, if you execute the following script, you'll get all services sorted by their status:

```
Get-Service  |  Sort-Object  -property  Status
```

You can also use a pipe to output text to a file using a script like the following:

```
"Hello, World!"  |  Out-File  C:\ps\test.txt
```

You can use multiple pipes. For instance, the following script lists all services, with the first pipe excluding stopped services and the second pipe limiting the list to display names only:

```
Get-Service  |  WHERE {$_.status -eq "Running"} | SELECT  displayname
# “$_.” defines current element in the pipe
```

2. Top 10 Active Directory Management Tasks with PowerShell

The easiest way to manage objects in an Active Directory domain is using the Active Directory Users and Computers (ADUC) MMC snap-in. However, what if you need to create multiple user accounts in bulk, or ADUC is not available for some reason? In this part, we'll explore how to perform most common AD management tasks with PowerShell.

Keep in mind that before you can work with Active Directory and its objects, you need to import the Active Directory module for Windows PowerShell. In Microsoft Windows Server 2008 R2, you need to enable this module by running the following command:

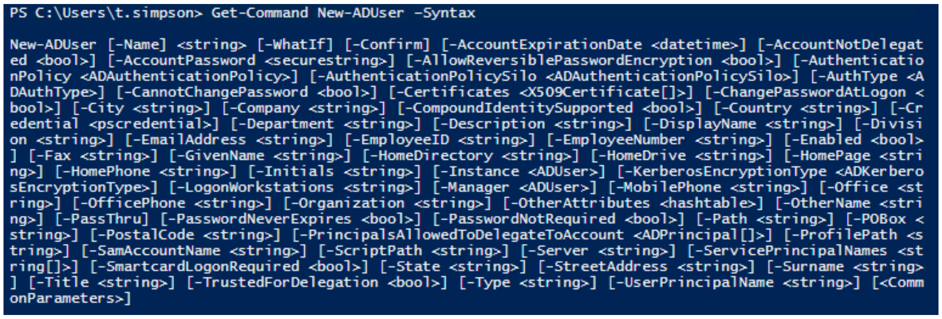
```
Import-Module  ActiveDirectory
```

In Microsoft Windows Server 2012 and later, this module is enabled by default.

2.1 Creating New User and Computer Accounts

You can create new user accounts in Active Directory using the cmdlet **New-ADUser**. You can get its full syntax by running the following command:

```
Get-Command  New-ADUser  –Syntax
```



```
PS C:\Users\t.simpson> Get-Command New-ADUser –Syntax

New-ADUser [-Name] <string> [-WhatIf] [-Confirm] [-AccountExpirationDate <datetime>] [-AccountNotDelegat
ed <bool>] [-AccountPassword <securestring>] [-AllowReversiblePasswordEncryption <bool>] [-Authenticatio
nPolicy <ADAuthenticationPolicy>] [-AuthenticationPolicySilo <ADAuthenticationPolicySilo>] [-AuthType <A
DAuthType>] [-CannotChangePassword <bool>] [-Certificates <X509Certificate[]>] [-ChangePasswordAtLogon <
bool>] [-City <string>] [-Company <string>] [-CompoundIdentitySupported <bool>] [-Country <string>] [-Cr
edential <pscredential>] [-Department <string>] [-Description <string>] [-DisplayName <string>] [-Divisi
on <string>] [-EmailAddress <string>] [-EmployeeID <string>] [-EmployeeNumber <string>] [-Enabled <bool>
] [-Fax <string>] [-GivenName <string>] [-HomeDirectory <string>] [-HomeDrive <string>] [-HomePage <stri
ng>] [-HomePhone <string>] [-Initials <string>] [-Instance <ADUser>] [-KerberosEncryptionType <ADKerbero
sEncryptionType>] [-LogonWorkstations <string>] [-Manager <ADUser>] [-MobilePhone <string>] [-Office <st
ring>] [-OfficePhone <string>] [-Organization <string>] [-OtherAttributes <hashtable>] [-OtherName <stri
ng>] [-PassThru] [-PasswordNeverExpires <bool>] [-PasswordNotRequired <bool>] [-Path <string>] [-POBox <
string>] [-PostalCode <string>] [-PrincipalsAllowedToDelegateToAccount <ADPrincipal[]>] [-ProfilePath <s
tring>] [-SamAccountName <string>] [-ScriptPath <string>] [-Server <string>] [-ServicePrincipalNames <st
ring[]>] [-SmartcardLogonRequired <bool>] [-State <string>] [-StreetAddress <string>] [-Surname <string>
] [-Title <string>] [-TrustedForDelegation <bool>] [-Type <string>] [-UserPrincipalName <string>] [<Comm
onParameters>]
```

When you know the syntax, it's easy to add users to Active Directory:

```
New-ADUser  B.Johnson
```

Accounts are created with the following default properties:

- Account is created in the “Users” container.
- Account is disabled.
- Account is a member of Domain Users group.
- No password is set.
- User must reset the password at the first logon.

Therefore, to make a new account that’s actually usable, you need to enable it using the **Enable-ADAccount** cmdlet and give it a password using the **Set-ADAccountPassword** cmdlet.

Let’s create a new account with the following attributes:

- **Name** — Jack Robinson
- **Given Name** — Jack
- **Surname** — Robinson
- **Account Name** — J.Robinson
- **User Principal Name** — J.Robinson@enterprise.com
- **Path** — “OU=Managers,DC=enterprise,DC=com”
- **Password Input** — Required
- **Status** — Enabled

Here’s the script we’ll use:

```
New-ADUser -Name "Jack Robinson" -GivenName "Jack" -Surname "Robinson" -SamAccountName
"J.Robinson" -UserPrincipalName "J.Robinson@enterprise.com" -Path
"OU=Managers,DC=enterprise,DC=com" -AccountPassword(Read-Host -AsSecureString "Input
Password") -Enabled $true
```

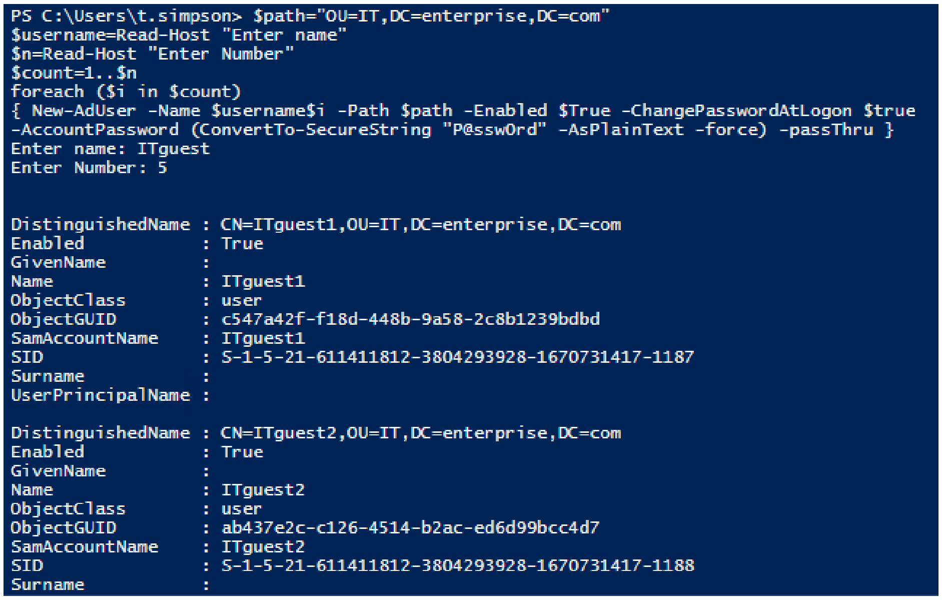
The **Read-Host** parameter will ask you to input new password. Note that the password should meet the length, complexity and history requirements of your domain security policy.

Now, let’s create ten similar Active Directory accounts in bulk and set a default password (P@ssw0rd) for each of them. To send the default password in a protected state, we must use the **ConvertTo-SecureString** parameter. Here’s the script to use:

```
parameter. Here’s the script to use:
$path="OU=IT,DC=enterprise,DC=com"
$username="ITclassuser"
$count=1..10
foreach ($i in $count)
{ New-AdUser -Name $username$i -Path $path -Enabled $True -ChangePasswordAtLogon $true  `
-AccountPassword (ConvertTo-SecureString "P@ssw0rd" -AsPlainText -force) -passThru }
```

To make the script more flexible, add the **Read-Host** parameter, which will ask for the number of users to be added:

```
$path="OU=IT,DC=enterprise,DC=com"
$username=Read-Host "Enter name"
$n=Read-Host "Enter Number"
$count=1..$n
foreach ($i in $count)
{ New-AdUser -Name $username$i -Path $path -Enabled $True -ChangePasswordAtLogon $true  `
-AccountPassword (ConvertTo-SecureString "P@ssw0rd" -AsPlainText -force) -passThru }
```

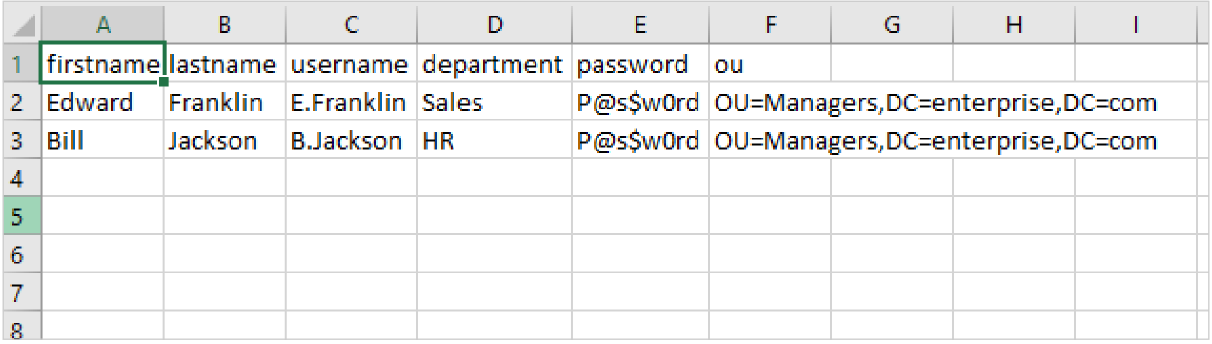


```
PS C:\Users\t.simpson> $path="OU=IT,DC=enterprise,DC=com"
$username=Read-Host "Enter name"
$n=Read-Host "Enter Number"
$count=1..$n
foreach ($i in $count)
{ New-AdUser -Name $username$i -Path $path -Enabled $True -ChangePasswordAtLogon $true
-AccountPassword (ConvertTo-SecureString "P@ssw0rd" -AsPlainText -force) -passThru }
Enter name: ITguest
Enter Number: 5


DistinguishedName : CN=ITguest1,OU=IT,DC=enterprise,DC=com
Enabled           : True
GivenName         :
Name              : ITguest1
ObjectClass       : user
ObjectGUID        : c547a42f-f18d-448b-9a58-2c8b1239bdbd
SamAccountName    : ITguest1
SID               : S-1-5-21-611411812-3804293928-1670731417-1187
Surname           :
UserPrincipalName :

DistinguishedName : CN=ITguest2,OU=IT,DC=enterprise,DC=com
Enabled           : True
GivenName         :
Name              : ITguest2
ObjectClass       : user
ObjectGUID        : ab437e2c-c126-4514-b2ac-ed6d99bcc4d7
SamAccountName    : ITguest2
SID               : S-1-5-21-611411812-3804293928-1670731417-1188
Surname           :
```

Another option for creating users in AD is to import them from a CSV file. This option is great when you have a list of users with predefined personal details such as:

- FirstName
- LastName
- Username
- Department
- Password
- OU

The CSV file must be in UTF8 encoding and contain contact data that looks like this:

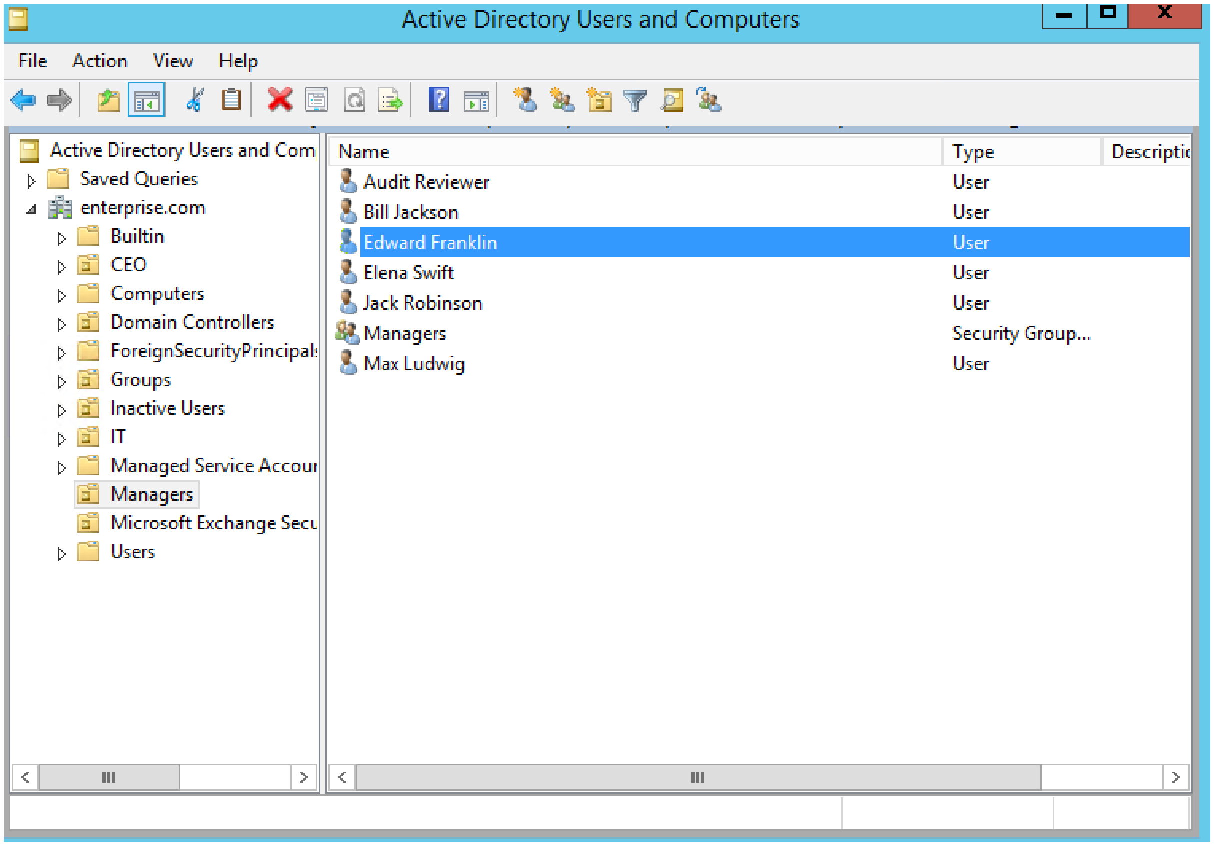


	A	B	C	D	E	F	G	H	I
1	firstname	lastname	username	department	password	ou			
2	Edward	Franklin	E.Franklin	Sales	P@s$w0rd	OU=Managers,DC=enterprise,DC=com			
3	Bill	Jackson	B.Jackson	HR	P@s$w0rd	OU=Managers,DC=enterprise,DC=com			
4									
5									
6									
7									
8									

The following script will create enabled user objects for any users in the CSV that don't already have accounts in AD. The "Reset password at the next logon" option will be enabled for the new accounts, so you can use your default password:

```
#Enter a path to your import CSV file
$ADUsers = Import-csv C:\scripts\newusers.csv

foreach ($User in $ADUsers)
{
        $Username          = $User.username
        $Password  = $User.password
        $Firstname = $User.firstname
        $Lastname  = $User.lastname
   $Department = $User.department
        $OU           = $User.ou

        #Check if the user account already exists in AD
        if (Get-ADUser -F {SamAccountName -eq $Username})
        {
                 #If user does exist, output a warning message
                 Write-Warning "A user account $Username has already exist in Active Directory."
        }
        else
        {
                #If a user does not exist then create a new user account

      #Account will be created in the OU listed in the $OU variable in the CSV file; don't forget
to change the domain name in the"-UserPrincipalName" variable
                New-ADUser `
         -SamAccountName $Username `
         -UserPrincipalName "$Username@yourdomain.com" `
         -Name "$Firstname $Lastname" `
         -GivenName $Firstname `
         -Surname $Lastname `
         -Enabled $True `
         -ChangePasswordAtLogon $True `
         -DisplayName "$Lastname, $Firstname" `
         -Department $Department `
         -Path $OU `
         -AccountPassword (convertto-securestring $Password -AsPlainText -Force)

        }
}
```

After script execution, we have two new users, Edward Franklin and Bill Jackson, in our Active Directory domain:



Name	Type	Descriptic
Audit Reviewer	User	
Bill Jackson	User	
Edward Franklin	User	
Elena Swift	User	
Jack Robinson	User	
Managers	Security Group...	
Max Ludwig	User	

To create a computer object, use the **New-ADComputer** cmdlet. For example, execute the following cmdlet parameters to create a computer object with “WKS932” as its name and the default LDAP path value:

```
New-ADComputer –Name “WKS932” –SamAccountName “WKS932”
```

If you have a list of computers that should be imported into Active Directory, save the list to a CSV file with the heading “computer” and the list of computer names in the column below it. Run the following PowerShell script on your domain controller to add computers from the CSV file, making sure you have the **Path** and **File** variables set correctly:

```
$File="C:\scripts\Computers.csv" # Specify the import CSV position.
$Path="OU=Devices,DC=enterprise,DC=com" # Specify the path to the OU.
Import-Csv -Path $File | ForEach-Object { New-ADComputer -Name $_.Computer
-Path $Path -Enabled $True}
```

2.2 Joining a Computer to a Domain and Removing a Computer from a Domain

Another common task is joining a computer to a domain controller. To join a PC to an Active Directory domain, run the following PowerShell script locally:

```
$dc = "ENTERPRISE" # Specify the domain to join.
$pw = "Password123" | ConvertTo-SecureString -asPlainText –Force # Specify the password for the domain admin.
$usr = "$dc\T.Simpson" # Specify the domain admin account.
$creds = New-Object System.Management.Automation.PSCredential($usr,$pw)
Add-Computer -DomainName $dc -Credential $creds -restart -force -verbose # Note that the computer will be restarted automatically.
```

The computer will restart and then join the domain; it will be added to the default container.

To join a computer to a DC remotely, you need to enhance this script this way:

```
$dc = "ENTERPRISE"
$pw = "Password123" | ConvertTo-SecureString -asPlainText -Force
$usr = "$dc\T.Simpson"
$pc = "R07GF" # Specify the computer that should be joined to the domain.
$creds = New-Object System.Management.Automation.PSCredential($usr,$pw)
Add-Computer -ComputerName $pc -LocalCredential $pc\admin -DomainName $dc -Credential $creds -Verbose -Restart -Force
```

The **$pc** variable and **–LocalCredential** parameter are used to authenticate the computer to the domain. Note that in order to use this method, you must disable the firewall on the local computer.

You can add more than one computer to the domain by either specifying them in the command line as a comma-delimited list or importing their names from a text file.

Here’s how to specify the computers in a comma-delimited list:

```
$dc = "ENTERPRISE"
$pw = "Password123" | ConvertTo-SecureString -asPlainText -Force
$usr = "$dc\T.Simpson"
$pc = "WKS034, WKS052, WKS057" # Specify the computers that should be joined to the domain.
$creds = New-Object System.Management.Automation.PSCredential($usr$pw)
Add-Computer -ComputerName $pc -LocalCredential $pc\admin -DomainName $dc -Credential $creds -Restart -Force
```

And here’s how to use a text file with the list of computers that should be joined:

```
$dc = "ENTERPRISE"
$pw = "Password123" | ConvertTo-SecureString -asPlainText -Force
$usr = "$dc\T.Simpson"
$pc = Get-Content -Path C:\Computers.txt # Specify the path to the computers list.
$creds = New-Object System.Management.Automation.PSCredential($usr,$pw)
Add-Computer -ComputerName $pc -LocalCredential $pc\admin -DomainName $dc -Credential
$creds -Restart -Force
```

To remove a computer from a domain remotely, use the **Remove-Computer** cmdlet. Here, we’re removing a computer from a domain, so no local credentials are needed and we can skip the **–LocalCredential** parameter:

```
$dc = "ENTERPRISE"
$pw = "Password123" | ConvertTo-SecureString -asPlainText -Force
$usr = "$dc\T.Simpson"
$pc = "R07GF"
$creds = New-Object System.Management.Automation.PSCredential($usr,$pw)
Remove-Computer -ComputerName $pc -Credential $creds –Verbose –Restart –Force
```

To remove multiple computers using a list in a TXT file, use the script above for joining computers to a DC, replacing the **Add-Computer** cmdlet with **Remove-Computer**. Note that you will still need domain admin credentials to complete this unjoin operation.

2.3 Renaming a Computer

To change a computer name, use the **Rename-Computer** cmdlet. Note that the computer must be online and connected to Active Directory.

```
Rename-Computer –ComputerName "FS1" -NewName "FS2"
```

If you want to run this script locally, it will look like this:

```
Rename-Computer -NewName "newname" -DomainCredential "Domain\Administrator"
```

You can improve the renaming script by joining the computer to the domain and putting it into the specified OU simultaneously. The script should be run on the target machine, not on the domain controller.

```
$NewComputerName = "Server3" # Specify the new computer name.
$DC = "contoso.com" # Specify the domain to join.
$Path = "OU=TestOU,DC=contoso,DC=com" # Specify the path to the OU where to put the computer account in the domain.
Add-Computer -DomainName $DC -OUPath $Path -NewName $NewComputerName –Restart –Force
```

The script will prompt for the credentials of an account that has permissions to join computers to the domain, and then the computer will be renamed, restarted and joined to the domain.

2.4 Resetting a Computer Account

Like a user account, a computer account interacts with Active Directory using a password. But for computer accounts, a password change is initiated every 30 days by default and the password is exempted from the domain's password policy. Password changes are driven by the client (computer), not AD.

Computer credentials usually unknown to the user because they are randomly set by the computer. But you can set your own password; here is a PowerShell script for doing so:

```
$pc = read-host –Prompt "Input computer name to reset" # Specify the computer name.
$pw = read-host –Prompt "Input random characters for temp password" –AsSecureString # Specify the password.
Get-ADComputer $pc | Set-ADAccountPassword –NewPassword:$pw -Reset:$true
```

2.5 Disabling User and Computer Accounts

To disable user, computer or service accounts, use the **Disable-ADAccount** cmdlet. The **-Identity** parameter specifies which account to disable. You can specify an account by its distinguished name, security identifier (SIDs), globally unique identifier (GUID) or Security Account Manager (SAM) account name.

```
Disable-AdAccount -Identity RussellS
```

If you specify a computer account name, remember to append a dollar sign ($) at the end of the name; otherwise, you'll get an error after script execution.

```
Disable-ADAccount -Identity fs1$
```

You can also disable accounts in bulk using a list in a text file:

```
$Pclist = Get-Content C:\scripts\Computer.txt # Specify the path to the computer list.
Foreach($pc in $Pclist)
{
Disable-ADAccount -Identity "$pc"
Get-ADComputer -Identity "$pc" | Move-ADObject -TargetPath “OU=Disabled
Computers,DC=enterprise,DC=com”
}
```

2.6 Deleting a Computer from Active Directory

To delete a computer account from AD, use the **Remove-ADObject** cmdlet:

```
Remove-ADObject -Identity "WKS932"
```

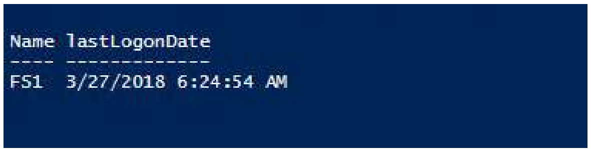
You will be prompted to confirm the deletion.

If you have a text file with a list of old computers, you can streamline the task of removing them using PowerShell. The following script will read the computer names from a TXT file and delete the corresponding accounts via a pipeline:

```
Get-Content C:\scripts\computersfordeletion.txt | % { Get-ADComputer -Filter { Name
-eq $_ } } | Remove-ADObject -Recursive
```

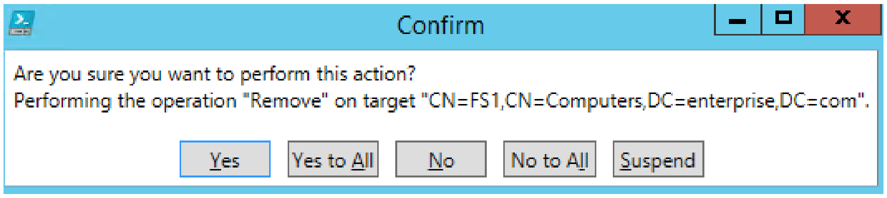
Stale accounts in Active Directory can be compromised, leading to security incidents, so it is critical to keep an eye on them. This PowerShell script will query Active Directory and return all computers that have not been logged in to for the past 30 days. It also will remove those accounts to keep your AD clean.

```
$stale = (Get-Date).AddDays(-30) # means 30 days since last logon; can be changed to any number.
Get-ADComputer -Property Name,lastLogonDate -Filter {lastLogonDate -lt $stale} |
FT Name,lastLogonDate
Get-ADComputer -Property Name,lastLogonDate -Filter {lastLogonDate -lt $stale} |
Remove-ADComputer
```



```
Name lastLogonDate
---- -------------
FS1  3/27/2018 6:24:54 AM
```

There is one computer, FS1, that has been not been logged on to for more than 30 days. The system will prompt for confirmation before deleting it from the domain:



Confirm

Are you sure you want to perform this action?
Performing the operation "Remove" on target "CN=FS1,CN=Computers,DC=enterprise,DC=com".

If you want to disable, rather than delete, the inactive computer accounts, replace the **Remove-ADComputer** cmdlet with **Set-ADComputer** and **-Enabled $false** parameter and value.

> Remember that it is critical to closely track all changes to computer accounts, so you can quickly spot any unwanted modifications and respond appropriately. Here's how to monitor computer account deletions.

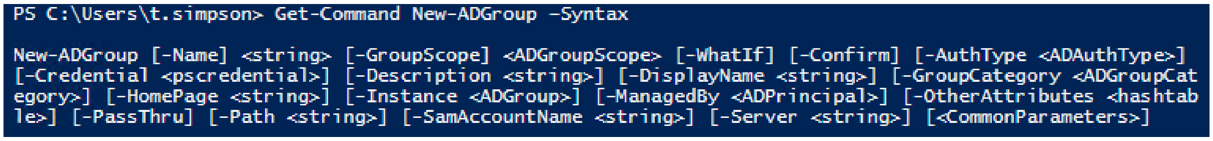
2.7 Creating and Deleting an Active Directory Group

In Active Directory, access to network resources is granted to security principals, such as user accounts and computer accounts, and those permissions can change over time. To simplify access management and improve security, medium and large companies often use Active Directory security groups, which can contain user and computer accounts as well as other groups. They also often use distribution groups to manage email distribution lists. Both security and distribution groups have unique SIDs and GUIDs.

> If you're not already familiar with AD groups and group management, please read the Active Directory Group Management Best Practice guide.

To create an AD group, use the **New-ADGroup** cmdlet. You can get its syntax by running the following command:

```
Get-Command New-ADGroup –Syntax
```



```
PS C:\Users\t.simpson> Get-Command New-ADGroup –Syntax

New-ADGroup [-Name] <string> [-GroupScope] <ADGroupScope> [-WhatIf] [-Confirm] [-AuthType <ADAuthType>]
[-Credential <pscredential>] [-Description <string>] [-DisplayName <string>] [-GroupCategory <ADGroupCat
egory>] [-HomePage <string>] [-Instance <ADGroup>] [-ManagedBy <ADPrincipal>] [-OtherAttributes <hashtab
le>] [-PassThru] [-Path <string>] [-SamAccountName <string>] [-Server <string>] [<CommonParameters>]
```

The easiest way to create a group is to run this short script:

```
New-ADGroup "Group Name"
```

The system will ask you to specify the **GroupScope** parameter and then it will create a new group. However, this group will have default values, such as:

- It will be created in the default LDAP container called "Users".
- It will have the "Security" group type.
- The Members, Member of, Description, Email and Notes fields will all be blank.

Let's create a security group called "Quality" in the "Production" OU (**-Path**); it should be a security group (**-GroupCategory**) and it should be global (**-GroupScope**):

```
New-ADGroup "Quality" -Path "OU=Production,DC=enterprise,dc=com" -GroupCategory
Security -GroupScope Global -PassThru –Verbose
```

If you want to make a universal distribution group, simply change the **–GroupCategory** parameter to "Distribution" and the **–GroupScope** parameter to "Universal". You can also change the LDAP path by changing the **–Path** parameter.

To delete an AD group, use the **Remove-ADGroup** cmdlet. The easiest script for that will look like this:

```
Remove-ADGroup -Identity Quality
```

You'll be prompted to confirm the deletion of the group.

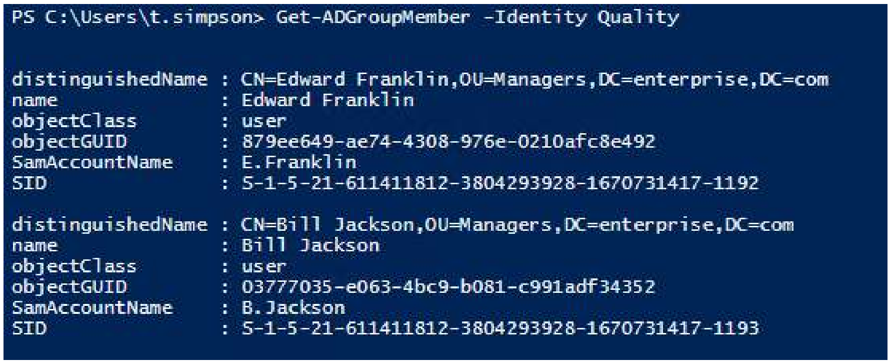
2.8 Adding Users and Computers to a Group

You can add users to an AD group with the **Add-AdGroupMember** cmdlet. For instance, if you needed to add two users, B.Jackson and E.Franklin, to the “Quality” group, here is what the script would look like:

```
Add-AdGroupMember -Identity Quality -Members B.Jackson, E.Franklin
```

Once you’ve added users to a security group, you can run this script to verify that they are listed as members:

```
Get-ADGroupMember -Identity Quality
```



```
PS C:\Users\t.simpson> Get-ADGroupMember -Identity Quality


distinguishedName : CN=Edward Franklin,OU=Managers,DC=enterprise,DC=com
name              : Edward Franklin
objectClass       : user
objectGUID        : 879ee649-ae74-4308-976e-0210afc8e492
SamAccountName    : E.Franklin
SID               : S-1-5-21-611411812-3804293928-1670731417-1192

distinguishedName : CN=Bill Jackson,OU=Managers,DC=enterprise,DC=com
name              : Bill Jackson
objectClass       : user
objectGUID        : 03777035-e063-4bc9-b081-c991adf34352
SamAccountName    : B.Jackson
SID               : S-1-5-21-611411812-3804293928-1670731417-1193
```

If you need to add users to another security or distribution group, such as Domain Admins, specify “Domain Admins” as the value for the **–Identit**y parameter. If you need one group to be a member of another, specify the group name as the value for the **–Members** parameter. The same principle applies to computer accounts, but you’ll need to append a dollar sign ($) to the end of the computer account name. For example, to add the computer “WKS043” to a group, specify “WKS043$” as the value for the **–Member** parameter:

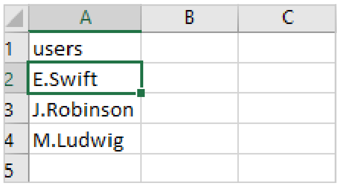
```
Add-AdGroupMember -Identity Quality -Members WKS043$
```

To add a user to multiple groups at once, run the following script.

```
"Managers","Quality" | Add-ADGroupMember -Members `
   (Read-Host -Prompt "Enter User Name")
```

You’ll be prompted to input the username.

If you want to add a large number of users to a group, you can specify them in a CSV file and then import that file. Note that the list of the usernames in the CSV file must contain the **SamAccountNames** in the “users” column, as shown below:



	A	B	C
1	users		
2	E.Swift		
3	J.Robinson		
4	M.Ludwig		
5			

To add users to group from a CSV file, run the following PowerShell script:

```
Import-CSV C:\scripts\users.csv -Header users | ForEach-Object {Add-AdGroupMember
-Identity "Quality" -members $_.users}
```

If you want to copy all members from one group to another group, run the following script:

```
Get-ADGroupMember “Quality” | Get-ADUser | ForEach-Object {Add-ADGroupMember
-Identity “QualityControl” -Members $_}
```

2.9 Removing Users and Computers from a Group

To remove a user from a group, use the **Remove-ADGroupMember** cmdlet:

```
Remove-ADGroupMember -Identity Quality -Members J.Robinson
```

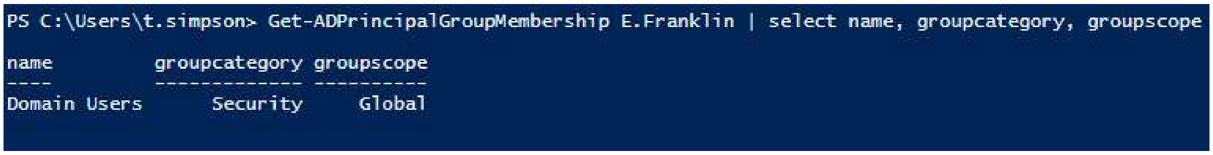
To remove a computer account from a group, specify the computer name with a dollar sign ($) at the end as the value for the **-Members** parameter.

An easy way to remove multiple users from an AD group is to create a CSV file with the list of usernames and then remove those users from the group object using this script:

```
Import-CSV C:\scripts\users.csv -Header users | ForEach-Object {Remove-ADGroupMember
-Identity "Quality" -members $_.users}
```

To remove a user from all groups, run this script:

```
Get-ADUser -Identity E.Franklin -Properties MemberOf | ForEach-Object {
  $_.MemberOf | Remove-ADGroupMember -Members $_.DistinguishedName -Confirm:$false
}
```



```
PS C:\Users\t.simpson> Get-ADPrincipalGroupMembership E.Franklin | select name, groupcategory, groupscope

name         groupcategory groupscope
----         ------------- ----------
Domain Users      Security     Global
```

Note that the user will lose all group membership except Domain Users, which can be removed manually if needed.

> Don’t forget to enable the Active Directory Recycle Bin feature so you can easily roll back your changes if something goes wrong.

2.10 Moving Users and Computers to a New Organizational Unit

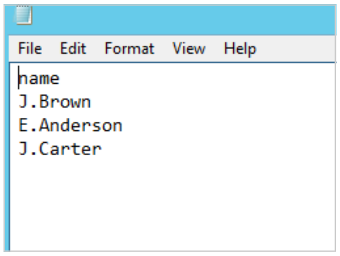
The PowerShell **Move-ADObject** cmdlet moves any object or set of objects (such as a user, a computer, a group or an OU) to a specified OU. The **-Identity** parameter specifies which Active Directory object or container to move. Note that you need to enter the full LDAP path or SID of the object; you cannot use its SamAccountName. Here’s how to move the user “John Brown” to the “Districts” OU:

```
Move-ADObject -Identity "CN=John Brown,CN=Users,DC=enterprise,DC=com" -TargetPath
"OU=Districts,OU=IT,DC=Enterprise,DC=Com"
```

Use the same syntax to move computer objects. The following command will move the computer “R07GF” to the “Computers” container:

```
Move-ADObject -Identity "CN=R07GF,OU=CEO,DC=enterprise,DC=com" -TargetPath
"CN=Computers,DC=Enterprise,DC=Com"
```

If you have a predefined list of objects to move, you can save it as a CSV file and then import that file to Active Directory. The CSV list should be in the following format:



```
name
J.Brown
E.Anderson
J.Carter
```

Use this PowerShell script to move AD user accounts listed in a CSV file:

```
# Specify target OU. This is where users will be moved.
$TargetOU =  "OU=Districts,OU=IT,DC=enterprise,DC=com"
# Specify CSV path. Import CSV file and assign it to a variable.
$Imported_csv = Import-Csv -Path "C:\temp\MoveList.csv"

$Imported_csv | ForEach-Object {
    # Retrieve DN of user.
    $UserDN  = (Get-ADUser -Identity $_.Name).distinguishedName
    # Move user to target OU.
    Move-ADObject  -Identity $UserDN  -TargetPath $TargetOU
}
```

To move AD computer accounts listed in a text file, use the following PowerShell script:

```
# Specify path to the text file with the computer account names.
$computers = Get-Content C:\Temp\Computers.txt

# Specify the path to the OU where computers will be moved.
$TargetOU   =  "OU=Districts,OU=IT,DC=enterprise,DC=com"
ForEach( $computer in $computers){
   Get-ADComputer $computer |
   Move-ADObject -TargetPath $TargetOU
}
```

3. Top 10 File System Management Tasks Using PowerShell

Every day, system administrators have to perform a range of standard operations on the numerous files and folders on their Windows servers, from managing user data on shared resources to maintaining backups properly. Using the following information, you can automate many of these tasks and save time for more important projects.

In this part, we explain how to automate file management and NTFS permissions management tasks with the help of PowerShell scripts.

3.1 Viewing Objects in a Directory

To view the content of a directory on a Windows file server, use the **Get-ChildItem** cmdlet. To show all hidden files, add the **-Force** parameter. The command below shows all root objects in the Shared folder:

```
Get-ChildItem -Force \\fs\Shared
```

If you want to also check all subfolders and their content, add the **-Recurse** parameter:

```
Get-ChildItem -Force \\fs\Shared -Recurse
```

To filter the output, add the **Filter, Exclude, Include** and **Path** parameters to the **Get-ChildItem** cmdlet. For advanced object filtering, use the **Where-Object** cmdlet. The script below searches for all executable files in the IT folder that were modified after April 1, 2018:

```
Get-ChildItem -Path \\fs\Shared\IT -Recurse -Include *.exe | Where-Object -FilterScript
{($_.LastWriteTime -gt '2018-04-01')}
```

3.2 Creating Files and Folders

To create new objects with Windows PowerShell, you can use the **New-Item** cmdlet and specify the type of item you want to create, such as a directory, file or registry key. For example, this command creates a folder:

```
New-Item -Path '\\fs\Shared\NewFolder' -ItemType Directory
```

And this command creates an empty file:

```
New-Item -Path '\\fs\Shared\NewFolder\newfile.txt' -ItemType File
```

If you need to create a file and write data to it, there are at least two built-in methods. The first is to use the **Out-File** cmdlet:

```
$text = 'Hello World!' | Out-File $text -FilePath C:\data\text.txt
```

To overwrite an existing file, use the –Force switch parameter.

Alternatively, you can create files using the **Export-Csv** cmdlet, which exports the output to a csv file that can be opened in Excel:

```
Get-ADuser -Filter * | Export-Csv -Path C:\data\ADusers.csv
```

3.3 Deleting Files and Folders

To delete objects, use the **Remove-Item** cmdlet. If the object is not empty, you'll be prompted to confirm the deletion. Here's how to delete the "IT" folder and all the subfolders and files inside it:

```
Remove-Item -Path '\\fs\shared\it\'
Confirm
The item at \\pdc\shared\it has children and the Recurse parameter was not specified. If you
continue, all children will be removed with the item. Are you sure you want to continue?
[Y] Yes  [A] Yes to All  [N] No  [L] No to All  [S] Suspend  [?] Help
(default is "Y"):
```

If you have already made sure that every object inside the folder should be deleted, you can use the **-Recurse** switch to skip the confirmation step:

```
Remove-Item -Path '\\fs\shared\it\' -Recurse
```

Sometimes you need to clean up old files from a certain directory. Here's the way to accomplish that:

```
$Folder = "C:\Backups"

#delete files older than 30 days
Get-ChildItem $Folder -Recurse -Force -ea 0 |
? {!$_.PsIsContainer -and $_.LastWriteTime -lt (Get-Date).AddDays(-30)} |
ForEach-Object {
        $_ | del -Force
        $_.FullName | Out-File C:\log\deletedbackups.txt -Append
}

#delete empty folders and subfolders if any exist
Get-ChildItem $Folder -Recurse -Force -ea 0 |
? {$_.PsIsContainer -eq $True} |
? {$_.getfiles().count -eq 0} |
ForEach-Object {
   $_ | del -Force
   $_.FullName | Out-File C:\log\deletedbackups.txt -Append
}
```

Here’s how to check whether a file exists and delete it if it does:

```
$FileName = 'C:\data\log.txt'
If (Test-Path $FileName){
        Remove-Item $FileName
}
```

To delete files from remote PCs, you must have the appropriate security permissions to access them. Be sure to use UNC paths so the script will correctly resolve the file locations.

```
$filelist = @(" \c$\Temp", "\c$\Backups") #variable to delete files and folder
$computerlist = Get-Content  C:\data\pc.txt #get list of remote pc's
   foreach ($computer in $computerlist){
      foreach ($file in $filelist){
         $filepath= Join-Path "\\$computer\" "$filelist" #generate unc paths to files or folders
         if (Test-Path $filepath)
         {
         Remove-Item $filepath -force -recurse -ErrorAction Continue}}}
```

3.4 Copying Files and Folders

The **Copy-Item** cmdlet enables you to copy objects from one path to another. The following command creates a backup by copying the file users.xlsx from one remote computer (fs) and saving it to another (fs2) over the network:

```
Copy-Item -Path \\fs\Shared\it\users.xlsx -Destination \\fs2\Backups\it\users.xlsx
```

If the target file already exists, the copy attempt will fail. To overwrite the existing file, even if it is in Read-Only mode, use the **-Force** parameter:

```
Copy-Item -Path \\fs\Shared\it\users.xlsx -Destination \\fs2\Backups\it\users.xlsx -Force
```

If you’re copying files to or from remote computers, be sure to use UNC paths. For example, use this command to copy files from a remote file server to the local C: directory:

```
Copy-Item \\fs\c$\temp -Recurse C:\data\
```

To copy files from your local directory to the remote folder, simply reverse the source and destination locations:

```
Copy-Item C:\data\ -Recurse \\fs\c$\temp
```

You can also copy files from one remote server to another. The following script recursively copies the \\fs\Shared\temp folder to \\fs\Shared\test:

```
Copy-Item \\fs\Shared\temp -Recurse \\fs\Shared\test
```

To copy only certain files from the source content to the destination, use the **-Filter** parameter. For instance, the following command copies only txt files from one folder to another:

```
Copy-Item -Filter *.txt -Path \\fs\Shared\it -Recurse -Destination \\fs2\Shared\text
```

You can also run the **XCOPY** and **ROBOCOPY** commands to copy files, or use COM objects as in the example below:

```
(New-Object -ComObject Scripting.FileSystemObject).CopyFile('\\fs\Shared', 'fs2\Backup')
```

3.5 Moving Files and Directories

The **Move-Item** cmdlet moves an item, including its properties, contents, and child items, from one location to another. It can also move a file or subdirectory from one directory to another location.

The following command moves a specific backup file from one location to another:

```
Move-Item -Path \\fs\Shared\Backups\1.bak -Destination \\fs2\Backups\archive\1.bak
```

This script moves the entire Backups folder and its content to another location:

```
Move-Item -Path \\fs\Shared\Backups -Destination \\fs2\Backups\archive
```

The Backups directory and all its files and subfolders will then appear in the archive directory.

3.6 Renaming Files

The **Rename-Item** cmdlet enables you to change the name of an object while leaving its content intact. It’s not possible to move items with the **Rename-Item** command; for that functionality, you should use the **Move-Item** cmdlet as described above.

The following command renames a file:

```
Rename-Item -Path "\\fs\Shared\temp.txt" -NewName "new_temp.txt"
```

To rename multiple files at once, use a script like this:

```
$files = Get-ChildItem -Path C:\Temp #create list of files
foreach ($file in $files)
{
    $newFileName=$file.Name.Replace("A","B")  #replace "A" with "B"
    Rename-Item $file $newFileName
}
```

3.7 Setting File and Folder Permissions

In order to implement a least-privilege model, which is a best practice for system security, IT security specialists and system administrators configure NTFS access control lists (ACLs) by adding access control entries (ACEs) on NTFS file servers. There are both basic and advanced NTFS permissions. You can set each of the permissions to “Allow” or “Deny”. You can find all these user permissions by running the following PowerShell script:

```
[system.enum]::getnames([System.Security.AccessControl.FileSystemRights])
```

> If you’re not familiar with NTFS permissions management, check out this NTFS Permissions Management Best Practice guide.

The PowerShell **set-acl** cmdlet is used to change the security descriptor of a specified item, such as a file, folder or a registry key; in other words, it is used to modify file or folder permissions. The following script sets the “FullControl” permission to “Allow” for the user “ENTERPRISE\T.Simpson” to the folder “Sales”:

```
$acl = Get-Acl \\fs1\shared\sales
$AccessRule = New-Object
System.Security.AccessControl.FileSystemAccessRule("ENTERPRISE\T.Simpson","FullControl","Allow")
$acl.SetAccessRule($AccessRule)
$acl | Set-Acl \\fs1\shared\sales
```

Note that the **SetAccessRule** parameter completely overwrites the permissions for a user or group, so you can change folder permissions using this parameter. If you just want to add permissions, use the **AddAccessRule** parameter instead. For instance, the following script adds the “FullControl” permission for the “ENTERPRISE\J.Carter” user account to the “Accounting” folder:

```
$acl = Get-Acl \\fs1\shared\Accounting
$AccessRule = New-Object
System.Security.AccessControl.FileSystemAccessRule("ENTERPRISE\J.Carter","FullControl","Allow")
$acl.AddAccessRule($AccessRule)
$acl | Set-Acl \\fs1\shared\Accounting
```

Here are the other permissions you can assign to users or security groups:

Access Right	Access Right’s Name in PowerShell
Full Control	FullControl
Traverse Folder / Execute File	ExecuteFile
List Folder / Read Data	ReadData
Read Attributes	ReadAttributes
Read Extended Attributes	ReadExtendedAttributes
Create Files / Write Data	CreateFiles
Create Folders / Append Data	AppendData
Write Attributes	WriteAttributes
Write Extended Attributes	WriteExtendedAttributes
Delete Subfolders and Files	DeleteSubdirectoriesAndFiles
Delete	Delete
Read Permissions	ReadPermissions
Change Permissions	ChangePermissions
Take Ownership	TakeOwnership

There are also sets of basic access rights that can be applied:

Access Rights Set	Rights Included in the Set	Name of the Set in PowerShell
Read	List Folder / Read Data Read Attributes Read Extended Attributes Read Permissions	Read
Write	Create Files / Write Data Create Folders / Append Data Write Attributes Write Extended Attributes	Write
Read and Execute	Traverse folder / Execute File List Folder / Read Data Read Attributes Read Extended Attributes Read Permissions	ReadAndExecute
Modify	Traverse folder / Execute File List Folder / Read Data Read Attributes Read Extended Attributes Create Files / Write Data Create Folders / Append Data Write Attributes Write Extended Attributes Delete Read Permissions	Modify

To copy permissions, a user must own both the source and target folders. The following command will copy the permissions from the “Accounting” folder to the “Sales” folder:

```
get-acl \\fs1\shared\accounting | Set-Acl \\fs1\shared\sales
```

> If you want to get a list of NTFS permissions via PowerShell, you can follow this easy how-to about exporting NTFS permissions to CSV.

3.8 Removing User Permissions

To remove permissions, use the **RemoveAccessRule** parameter. Let’s delete the “Allow FullControl” permission for T.Simpson to the “Sales” folder:

```
$acl = Get-Acl \\fs1\shared\sales
$AccessRule = New-Object
System.Security.AccessControl.FileSystemAccessRule("ENTERPRISE\T.Simpson","FullControl","Allow")
$acl.RemoveAccessRule($AccessRule)
$acl | Set-Acl \\fs1\shared\sales
```

Note that **RemoveAccessRule** deletes only specific permissions. To completely wipe T.Simpson’s permissions to the “Sales” folder, use the **PurgeAccessRules** command:

```
$acl = Get-Acl \\fs1\shared\sales
$usersid = New-Object System.Security.Principal.Ntaccount ("ENTERPRISE\T.Simpson")
$acl.PurgeAccessRules($usersid)
$acl | Set-Acl \\fs1\shared\sales
```

Note that **PurgeAccessRules** doesn’t work with a string user name; it works only with SIDs. Therefore, we used the “Ntaccount” class to convert the user account name from a string into a SID. Also note that **PurgeAccessRules** works only with explicit permissions; it does not purge inherited ones.

3.9 Enabling and Disabling Permissions Inheritance

NTFS permissions can be either explicit or inherited. Explicit permissions are permissions that are configured individually, while inherited permissions are inherited from the parent folder. The hierarchy for permissions is as follows:

- Explicit Deny
- Explicit Allow
- Inherited Deny
- Inherited Allow

To manage inheritance, we use the **SetAccessRuleProtection** method. It has two parameters:

- The first parameter is responsible for blocking inheritance from the parent folder. It has two states: "$true" and "$false".
- The second parameter determines whether the current inherited permissions are retained or removed. It has the same two states: "$true" and "$false".

Let's disable inheritance for the "Sales" folder and delete all inherited permissions as well:

```
$acl = Get-Acl \\fs1\shared\sales
$acl.SetAccessRuleProtection($true,$false)
$acl | Set-Acl \\fs1\shared\sales
```

All inherited permissions were removed; only access permissions added explicitly are left.

Let's revert this change and re-enable inheritance for the "Sales" folder:

```
$acl = Get-Acl \\fs1\shared\sales
$acl.SetAccessRuleProtection($false,$true)
$acl | Set-Acl \\fs1\shared\sales
```

3.10 Changing File and Folder Ownership

If you want to set an owner for a folder, you need to run the **SetOwner** method. Let's make "ENTERPRISE\J.Carter" the owner of the "Sales" folder:

```
$acl = Get-Acl \\fs1\shared\sales
$object = New-Object System.Security.Principal.Ntaccount("ENTERPRISE\J.Carter")
$acl.SetOwner($object)
$acl | Set-Acl \\fs1\shared\sales
```

Notice that we again used the **Ntaccount** class to convert the user account name from a string into a SID.

Note that the **SetOwner** method does not enable you to change the owner to any account you want; the account must have the "Take Ownership", "Read" and "Change Permissions" rights.

4. Automating PowerShell Scripts

Now let’s explore you how to create scheduled tasks using PowerShell scripts and Microsoft Windows Task Scheduler.

4.1 Creating Scheduled Tasks with PowerShell Scripts

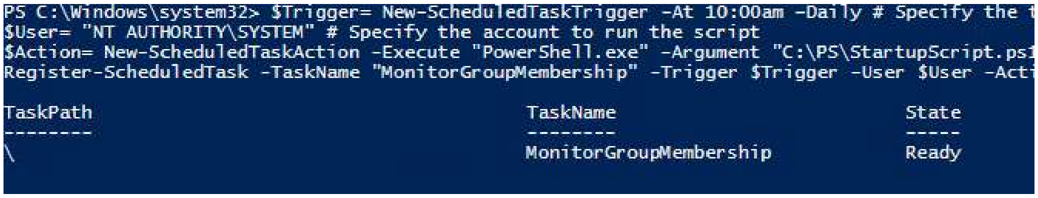
Suppose that each day at 10 AM, we want to execute a PowerShell script that monitors changes to group membership in an Active Directory site.

In Windows Powershell 2.0 (Windows 7 or Windows Server 2008 R2), to create a scheduled job, you must use the **Task Scheduler** module. Install the module by running the **Import-Module TaskScheduler** command, and then use the following script to create a task that will execute the PowerShell script named “GroupMembershipChanges.ps1” daily at 10 AM:

```
Import-Module TaskScheduler $task = New-Task
$task.Settings.Hidden = $true
Add-TaskAction -Task $task -Path C:\Windows\system32\WindowsPowerShell\v1.0\powershell.exe
–Arguments “-File C:\Scripts\GroupMembershipChanges.ps1”
Add-TaskTrigger -Task $task -Daily -At “10:00”
Register-ScheduledJob –Name ”Monitor Group Management” -Task $task
```

Windows PowerShell 3.0 and 4.0 (Windows Server 2012 R2 and above) don’t include the Task Scheduler module, so this script will not work. Instead, PowerShell 3.0 and 4.0 include new cmdlets for creating scheduled tasks, **New-ScheduledTaskTrigger** and **Register-ScheduledTask**, which make creating a scheduled task much easier and more convenient. So let’s create a task that will execute our script daily at 10 AM using the system account (SYSTEM), which has elevated privileges:

```
$Trigger= New-ScheduledTaskTrigger -At 10:00am –Daily # Specify the trigger settings
$User= "NT AUTHORITY\SYSTEM" # Specify the account to run the script
$Action= New-ScheduledTaskAction -Execute "PowerShell.exe" -Argument "C:\PS\StartupScript.ps1"
# Specify what program to run and its parameters
Register-ScheduledTask -TaskName "MonitorGroupMembership" -Trigger $Trigger -User $User
-Action $Action -RunLevel Highest –Force # Specify the name of the task
```



```
PS C:\Windows\system32> $Trigger= New-ScheduledTaskTrigger -At 10:00am -Daily # Specify the t
$User= "NT AUTHORITY\SYSTEM" # Specify the account to run the script
$Action= New-ScheduledTaskAction -Execute "PowerShell.exe" -Argument "C:\PS\StartupScript.ps1
Register-ScheduledTask -TaskName "MonitorGroupMembership" -Trigger $Trigger -User $User -Acti

TaskPath                                       TaskName                          State
--------                                       --------                          -----
\                                              MonitorGroupMembership            Ready
```

Other trigger options that could be useful in creating new tasks include:

- **-AtStartup** — Triggers the task at Windows startup.
- **-AtLogon** — Triggers the task when the user signs in.
- **-Once** — Triggers the task once. You can set a repetition interval using the –RepetitionInterval parameter.
- **-Weekly** — Triggers the task once a week.

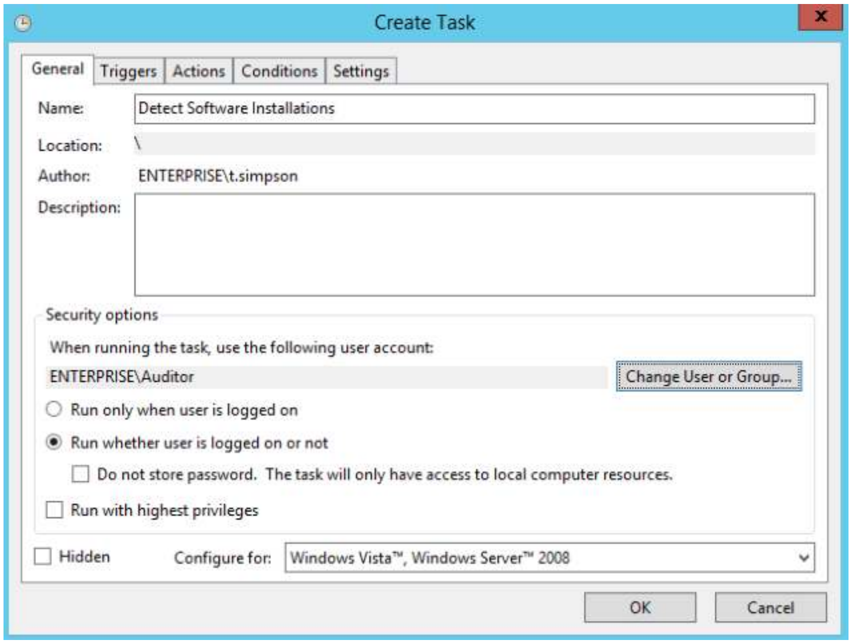
Note that, it is not possible to trigger execution “on an event” using these cmdlets; PowerShell scripts with “on an event” triggers are much more complicated. However, it is possible to do so with the Task Scheduler tool, so this is a real disadvantage of using PowerShell rather than Task Scheduler.

4.2 Running PowerShell Scripts with Task Scheduler

Task Scheduler can help you automatically launch a program or PowerShell script at a certain time or when certain conditions are met. You can also schedule sending emails and even displaying certain messages.

To create a task, open Task Scheduler by pressing “Windows+R” and typing **taskschd.msc** in the window that opens. Then take the following steps:

1. Click **Create a task** and enter a name and description for the new task. To run the program with administrator privileges, check the **Run with the highest privileges** box. In our example, we’ll assign a service account to run the task and run it regardless of whether the user is logged on.



Name: Detect Software Installations

Location: \

Author: ENTERPRISE\t.simpson

Description:

Security options

When running the task, use the following user account:

ENTERPRISE\Auditor

○ Run only when user is logged on

● Run whether user is logged on or not

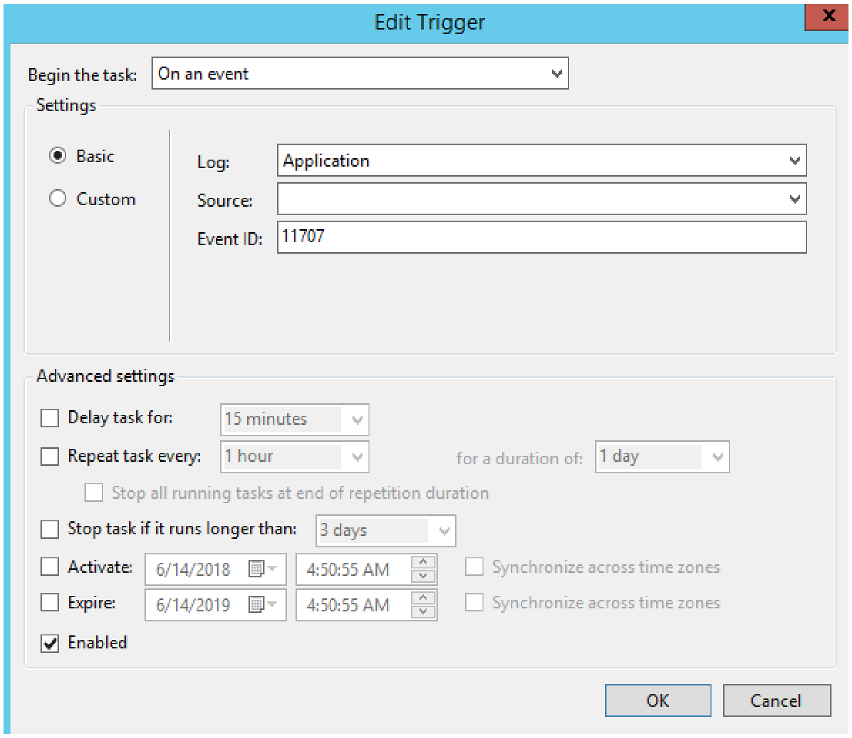
□ Do not store password. The task will only have access to local computer resources.

□ Run with highest privileges

□ Hidden

Configure for: Windows Vista™, Windows Server™ 2008

2. Switch to the **Triggers** tab and click the **New**… button. Here you can specify the conditions that trigger the task to be executed. For example, you can have it executed on schedule, at logon, on idle, at startup or whenever a particular event occurs. We want our task to be triggered by any new software installation, so we choose **On an event** from the drop-down menu and select **Application** from the **Log** drop-down. Leave the **Source** parameter blank and enter “11707” for the **EventID**. Click **OK** to save your changes.



Edit Trigger

Begin the task: On an event

Settings

● Basic
○ Custom

Log: Application
Source:
Event ID: 11707

Advanced settings

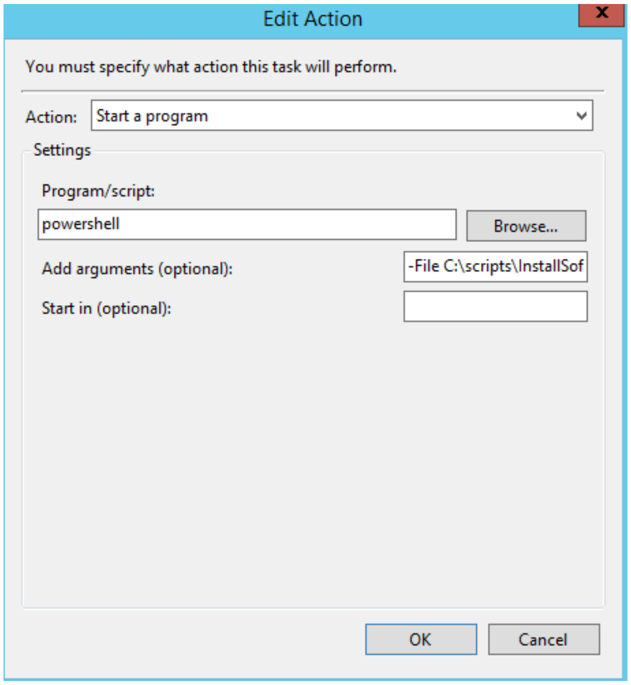
□ Delay task for: 15 minutes
□ Repeat task every: 1 hour   for a duration of: 1 day
□ Stop all running tasks at end of repetition duration
□ Stop task if it runs longer than: 3 days
□ Activate: 6/14/2018 4:50:55 AM □ Synchronize across time zones
□ Expire: 6/14/2019 4:50:55 AM □ Synchronize across time zones
☑ Enabled

3. Navigate to the **Actions** tab and click **New**.... Here you can specify the actions that will be executed whenever the trigger conditions are met. For instance, you can send an email or display a message. In our case, we want to start a program. You can find the script here; it will send an alert with the event details about the installed software.

To schedule the PowerShell script, specify the following parameters:

- **Action** — Start a program
- **Program\script** — powershell
- **Add arguments (optional)** — -File [Specify the file path to the script here]

Click **OK** to save your changes.



Edit Action

You must specify what action this task will perform.

Action: Start a program

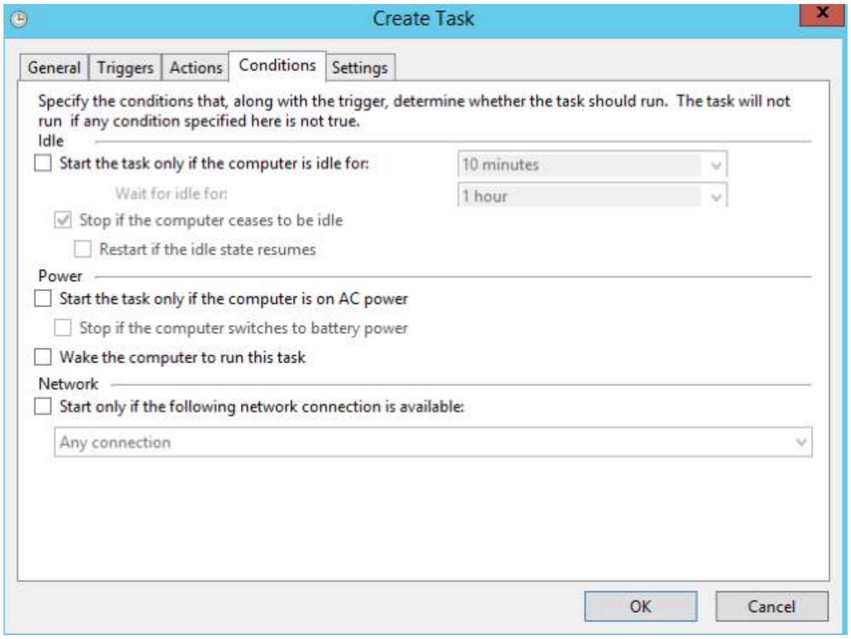
Settings

Program/script: powershell

Add arguments (optional): -File C:\scripts\InstallSof

Start in (optional):

4. The **Conditions** tab enables you to specify the conditions that, along with the trigger, determine whether the task should be run. In our case, we should leave the default settings on this tab.



Specify the conditions that, along with the trigger, determine whether the task should run. The task will not run if any condition specified here is not true.

Idle

□ Start the task only if the computer is idle for: 10 minutes

Wait for idle for: 1 hour

☑ Stop if the computer ceases to be idle

□ Restart if the idle state resumes

Power

□ Start the task only if the computer is on AC power

□ Stop if the computer switches to battery power

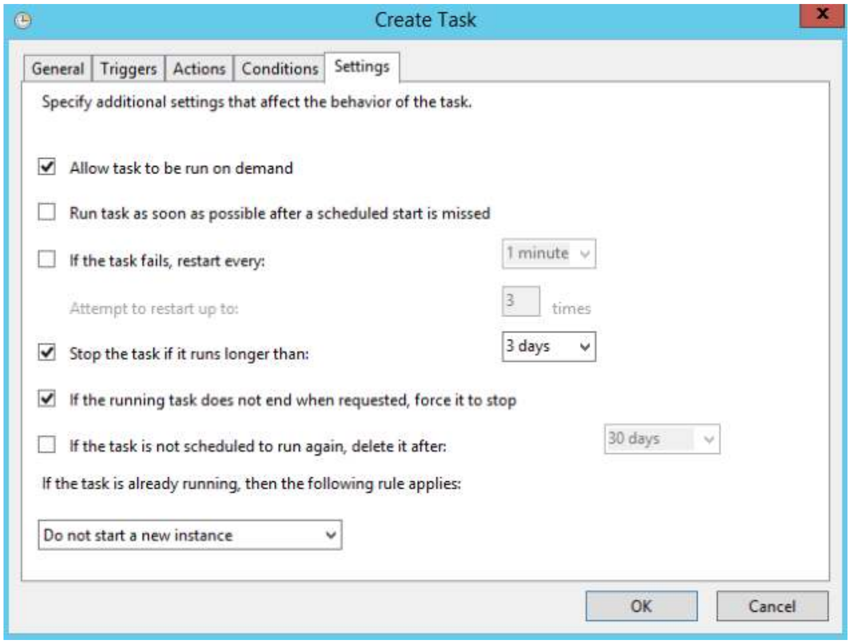
□ Wake the computer to run this task

Network

□ Start only if the following network connection is available:

Any connection

5. You can also set up additional parameters for your scheduled task on the **Settings** tab. For our example, though, we'll leave them unchanged.



Specify additional settings that affect the behavior of the task.

☑ Allow task to be run on demand

□ Run task as soon as possible after a scheduled start is missed

□ If the task fails, restart every: 1 minute

Attempt to restart up to: 3 times

☑ Stop the task if it runs longer than: 3 days

☑ If the running task does not end when requested, force it to stop

□ If the task is not scheduled to run again, delete it after: 30 days

If the task is already running, then the following rule applies:

Do not start a new instance

6. When the task is completely set up, the system will ask you for the service account password. Note that this account must have the “Log on as Batch Job” right. Enter the password and click **OK** to save the task.

7. For Task Scheduler to function properly, the Job Scheduler service must be set to start automatically. Run **Services.msc**. In the list of services, find Task Scheduler and double-click it. On the **General** tab, set the startup type to “Automatic” and click **OK** to save your change.

Now whenever new software is installed on your Microsoft Windows Server, you will be notified via an email that details the time of the installation, the name of the software and the user ID (SID) of the person who installed it.

To modify an existing task, right-click it in the list, select **Properties**, edit the required settings and click **OK**. To delete a scheduled task, right-click it, select **Delete** and confirm the action.

Useful References

Blog post

Introduction to PowerShell
PowerShell Variables and Arrays
How to Automate PowerShell Scripts with Task Scheduler
How to Get, Edit, Create and Delete Registry Keys with PowerShell
PowerShell File Management
How to Create, Change and Test Passwords Using PowerShell
How to lock, unlock, enable and disable AD accounts with PowerShell
How to Add and Remove AD Groups and Objects in Groups with PowerShell
How to Create New Active Directory Users with PowerShell
How to Collect AD Site Information Using PowerShell
Create AD Users in Bulk and Email Their Credentials Using PowerShell
Visibility Tips: Who Has Access to Mailbox?
How to Create, Delete, Rename, Disable and Join Computers in AD Using PowerShell
How to Add, Delete and Change Local Users and Groups with PowerShell
How to Create, Assign Licenses to, Disable and Delete Office 365 Accounts
Ten Most Useful Office 365 PowerShell Commands
PowerShell: Find Disabled or Inactive Users and Computers in AD
Using PowerShell to Create ZIP Archives and Unzip Files
Managing SharePoint Online using PowerShell
Most Useful SharePoint PowerShell Commands

How-to

How to Monitor Folder for New Files
How to Find Account Lockout Source
How to Get User Login
How to Find Locked Out User Accounts in AD
How to Find Expired Accounts in Active Directory
How to Find Inactive Computers in Active Directory
How to Get Local Administrators
How to Get Disabled Users
How to Get Local Users
How to Count Files in a Directory
How to Get ACL for a Folder
How to Get a List of All Users from a Specific OU
How to List All Computers in an OU

How-to	How to Get a List of Users with Password Never Expires
	How to Collect Server Inventory
	How to Find Duplicate Files
	How to List All User Accounts on a Windows System
Webinar	Top 5 Security Tasks to Automate with PowerShell
	PowerShell Essentials for Bolstering AD and Azure AD Security
	Managing SharePoint Online and Exchange Online with PowerShell
	Rapid Incident Response in Office 365 with PowerShell
	4 Handy PowerShell Commands for Managing the File System
	Most Useful PowerShell Cmdlets for Managing and Securing Active Directory

Increase the efficiency of administration tasks in your Windows environment with Netwrix Auditor

- Get complete details on all changes made across your critical Windows systems to make it quick and easy to spot and revert improper or malicious actions.
- Automate regular auditing of critical Windows systems to save time while ensuring continuous availability and minimizing business disruptions.
- Super-charge your efficiency with human-readable reports that eliminate time-consuming manual analysis of mountains of cryptic log data.

Download Free 20-Day Trial

About Netwrix

Netwrix is a software company that enables information security and governance professionals to reclaim control over sensitive, regulated and business-critical data, regardless of where it resides. Over 10,000 organizations worldwide rely on Netwrix solutions to secure sensitive data, realize the full business value of enterprise content, pass compliance audits with less effort and expense, and increase the productivity of IT teams and knowledge workers.

Founded in 2006, Netwrix has earned more than 150 industry awards and been named to both the Inc. 5000 and Deloitte Technology Fast 500 lists of the fastest growing companies in the U.S.

For more information, visit www.netwrix.com.

Next Steps

Free trial – Set up Netwrix in your own test environment: netwrix.com/freetrial

In-Browser Demo – See the unified platform in action, no deployment required: netwrix.com/browser_demo

Live Demo – Take a product tour with a Netwrix expert: netwrix.com/livedemo

Request Quote – Receive pricing information: netwrix.com/buy

CORPORATE HEADQUARTER:	PHONES:
300 Spectrum Center Drive Suite 200 Irvine, CA 92618	1-949-407-51 Toll-free (USA): 888-638-9749
565 Metro Place S, Suite 400 Dublin, OH 43017	1-201-490-8840
5 New Street Square London EC4A 3TW	+44 (0) 203 588 3023

OTHER LOCATIONS:

Spain:	+34 911 982608
Netherlands:	+31 858 887 804
Sweden:	+46 8 525 03487
Switzerland:	+41 43 508 3472
France:	+33 9 75 18 11 19
Germany:	+49 711 899 89 187
Hong Kong:	+852 5808 1306
Italy:	+39 02 947 53539

SOCIAL:

netwrix.com/social