3

Managing Windows Active Directory

In this chapter, we cover the following recipes:

* Installing Active Directory with DNS
* Creating and managing AD users, groups, and computers
* Adding users to AD via a CSV file
* Creating a group policy object
* Reporting on AD users
* Finding expired computers and disabled users in AD

# Introduction

A core component of almost all organizations' IT infrastructure is Active Directory (AD). Active Directory provides access control, user and system customization, and a wealth of directory and other services.

Using AD Domain Services, you can deploy a series of domain controllers throughout your organization. Use the Installing Active Directory and DNS recipe to install a pair of domain controllers that also provides a DNS service for your newly created AD forest. In the Creating and managing AD users recipe, you create, move, and remove user objects, as well as creating OUs and groups and establishing membership of these AD objects. In the Adding users to AD via a CSV recipe, you use a comma-separated value file containing details of users you wish to add to the AD.

The group policy is another important feature of Active Directory. With the group policy, you can define policies for users and computers that are applied automatically to the user and/or computer. In the Creating a group policy object recipe, you create a simple GPO and observe applying that policy.

In the penultimate recipe, Reporting on AD users, you examine the AD to find details on users who haven't logged on for a while, computers that have not been used for a while, and users who are members of special security groups (such as enterprise administrators). The final recipe, Finding expired computers and disabled users, finds computer and user objects that have not been used in a while. This can help you to keep your AD free of stale objects.

# Installing Active Directory with DNS

Installing Active Directory and DNS has always been fairly straightforward. You can always use Server Manager, but using PowerShell is really quite simple.

## Getting ready

This recipe starts with two non-domain joined hosts, DC1 and DC2. Each host is running Windows Server 2019 with no tools loaded. After creating the initial forest and forest root server (DC1), you convert DC2 to be another domain controller that also runs DNS.

## How to do it...

1. On DC1, install the AD Domain Services feature and the associated management tools:

Install-WindowsFeature AD-Domain-Services -IncludeManagementTools

1. Install DC1 as the forest root domain controller in the DC1.Reskit.Org forest:

$PSSHT = @{

String = 'Pa$$w0rd'

AsPlainText = $true

Force = $true

}

$PSS = ConvertTo-SecureString @PSSHT

$ADHT = @{

DomainName = 'Reskit.Org'

SafeModeAdministratorPassword = $PSS

InstallDNS = $true

DomainMode = 'WinThreshold'

ForestMode = 'WinThreshold'

Force = $true

NoRebootOnCompletion = $true

}

Install-ADDSForest @ADHT

1. Restart the DC1 computer:

Restart-Computer -Force

1. After rebooting, log back in to DC1 as Reskit\Administrator, then view the RootDSE entry on DC1:

Get-ADRootDSE |

Format-Table -Property DNS\*, \*Functionality

The next part of this recipe runs on DC2. Based on the previous steps, DC1 is now a domain controller. DC2 begins as a workgroup server with no additional roles/features added:

1. Log on to DC2 and check that DC1 can be resolved and can be reached over 445 and 389 from DC2:

Resolve-DnsName -Name DC1.Reskit.Org -Server DC1 -Type A

Test-NetConnection -ComputerName DC1.Reskit.Org -Port 445

Test-NetConnection -ComputerName DC1.Reskit.Org -Port 389

1. Add the AD DS features to DC2:

$Features = 'AD-Domain-Services', 'DNS','RSAT-DHCP',

'Web-Mgmt-Tools'

Install-WindowsFeature -Name $Features

1. Promote DC2 to be an additional domain controller in the Reskit.Org domain:

$URK = "Administrator@Reskit.Org"

$PSSHT = @{

String = 'Pa$$w0rd'

AsPlainText = $true

Force = $true

}

$PSS = ConvertTo-SecureString @pssht

$CREDHT = @{

Typename = 'System.Management.Automation.PSCredential'

ArgumentList = "$URK,$PSS"

}

$CredRK = New-Object @CREDHT

$IHT =@{

DomainName = 'Reskit.org'

SafeModeAdministratorPassword = $PSS

SiteName = 'Default-First-Site-Name'

NoRebootOnCompletion = $true

Force = $true

}

Install-ADDSDomainController @IHT -Credential $CredRK

1. Reboot the DC2 host:

Restart-Computer -Force

1. After rebooting, log on to DC1 and view the forest:

Get-AdForest |

Format-Table -Property \*master\*,global\*,Domains

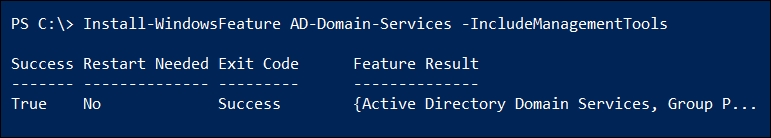
1. View the details of the Reskit.Org domain:

Get-ADDomain |

Format-Table -Property DNS\*,PDC\*,\*Master,Replica\*

## How it works...

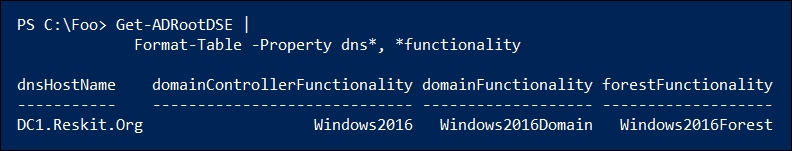
In step 1, you install the AD Domain Services feature and the management tools (the PowerShell module and AD-related MMC consoles), which looks like this:



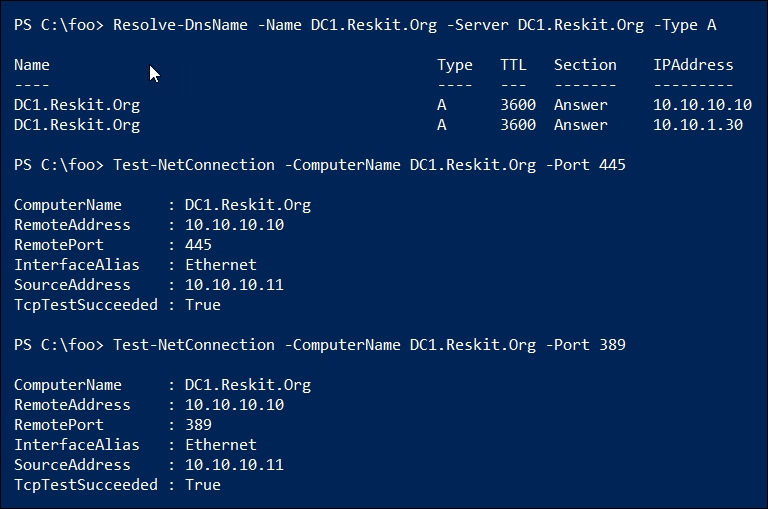
In step 2, you install DC1 as the forest root domain controller, which looks like this:



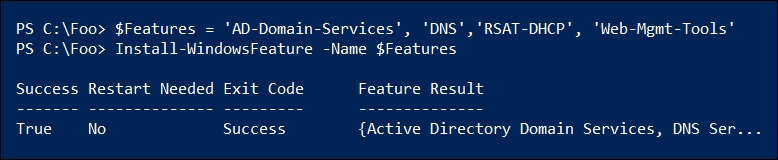
This step generates several warning messages. In this case, these warnings are benign and you can ignore them. After the DC promotion has completed, in step 3, you reboot the host. This generates no console output. Once you have rebooted DC1, in step 4, after you log on to DC1, you examine the RootDSE, which looks like this:



In step 5, after logging in to DC2, you check to ensure that you can resolve the IP address for DC1 from DC2 and that you can reach the other DC over ports 445 and 389. If these checks fail, promoting DC2 to be a domain controller is also going to fail. The output of this step looks like this:

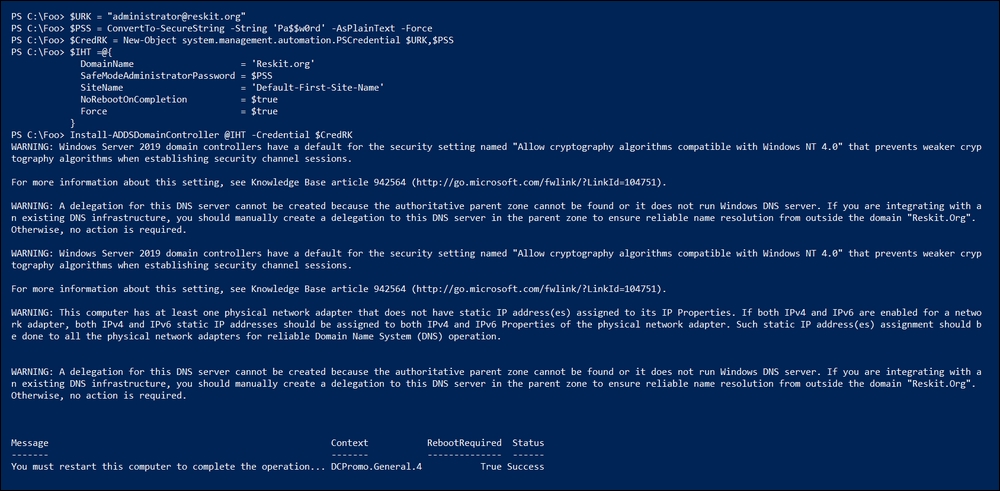


These tests show that DC2 can contact DC1 over key ports, so should be capable of being promoted to be a domain controller. In step 6, you add the ADDS features to DC2, as you did earlier for DC1. The output of this step looks like this:



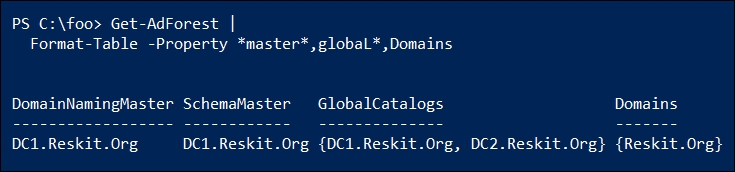
In this step, you add some additional tools, including the RSAT DHCP tools. You have options as to how much you add at each point. In this case, you need the AD-Domain-Services and DNS Services features, whilst the others are optional.

With connectivity tests succeeding and the pre-requisites installed, in step 7, you promote DC2 to be another domain controller in the Reskit.Org domain, like this:

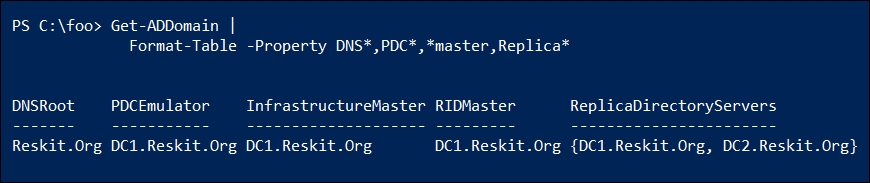


After completing the installation of DC2 as a domain controller, in step 8, you reboot the host which produces no output.

After DC2 has completed rebooting in step 9, log in and examine aspects of the Reskit.Org forest, like this:



In the final step, step 10, you examine details about the Reskit.Org domain, like this:



## There's more...

In step 2, you create DC1 as the first domain controller in the Reskit.Org forest. After the installation process completes, you must reboot DC1 before it can function as a DC, which you do in step 3.

In step 4, you examine the Root Directory Server Agent Service Entry or RootDSE in your domain. This entry, which is part of the LDAP standard as defined in RFC 2251 Section 3.4, enables an LDAP server to provide information about the capabilities of that server and the data that it contains to other LDAP servers and clients. This information is available without requiring any authentication. For more details on the RootDSE object and its attributes, see: <https://docs.microsoft.com/windows/desktop/adschema/rootdse>.

In step 5, you checked for connectivity on ports 445 and 389. With Windows Server 2019, port 445 is used for SMB file sharing, while port 389 is the port for LDAP. Domain-joined systems need access to these ports to access the domain controller for group policy details.

After completing the installation of AD on DC2, you need to reboot DC2, after which DC2 is a second domain controller in the Reskit.Org domain and is also a DNS server.

# Creating and managing AD users, groups, and computers

Once you have created your forest/domain and your domain controllers, you can begin to manage the core objects in AD, namely, users, groups, and computers and organizational units (OUs).

User and computer accounts identify a specific user or computer. These objects are used to enable the computer and the user to log on securely. Groups enable you to collect users into a single (group) account that simplifies the setting up of access controls on resources such as files or file shares. OUs enable you to partition users, computers, and groups into separate containers.

OUs serve two important roles in your AD. The first is role delegation. You can delegate the management of any OU (and child OUs) to be carried out by different groups. For example, you could create a top-level OU called UK in the Reskit.Org domain. You could then delegate permissions to the objects in this OU to a group, such as UKAdmins, enabling a member of that group to manage AD objects in and below the UK OU. Another OU, for example NA, could be delegated to a separate group, such as the North America Admins group (for example, NAAdmins). This enables you to delegate management.

The second role played by OUs is to act as a target for group policy objects. You could create a group policy object for the IT team and apply it to the IT OU. You could create a separate OU and create GPOs that apply to only the computer and user objects in that OU. Thus, each user and computer in a given OU are configured based on the GPO.

In this recipe, you create, update, and remove AD user objects as well as creating an OU and a security group, which you also populate. This recipe only creates and manages the AD objects. You assign a group policy in a later recipe, Creating a group policy object.

This recipe creates objects that are used in other recipes in this book.

Getting ready

This recipe assumes you have the Reskit.Org domain created, as performed in the Install Active Directory with DNS recipe, and you have two working domain controllers (DC1 and DC2).

Run this recipe to create and manage OUs, users, computers, and groups on DC1. Once you have created the various objects in this recipe on one DC, AD replication replicates those updates to the other DC, DC2.

## How to do it...

1. Create a hash table for general user attributes:

$PW = 'Pa$$w0rd'

$PSS = ConvertTo-SecureString -String $PW -AsPlainText -Force

$NewUserHT = @{}

$NewUserHT.AccountPassword = $PSS

$NewUserHT.Enabled = $true

$NewUserHT.PasswordNeverExpires = $true

$NewUserHT.ChangePasswordAtLogon = $false

1. Create two new users, utilizing the hash table created in the previous step:

# Create the first user ThomasL

$NewUserHT.SamAccountName = 'ThomasL'

$NewUserHT.UserPrincipalName = 'thomasL@reskit.org'

$NewUserHT.Name = 'ThomasL'

$NewUserHT.DisplayName = 'Thomas Lee (IT)'

New-ADUser @NewUserHT

# Create a second user RLT

$NewUserHT.SamAccountName = 'RLT'

$NewUserHT.UserPrincipalName = 'rlt@reskit.org'

$NewUserHT.Name = 'Rebecca Tanner'

$NewUserHT.DisplayName = 'Rebecca Tanner (IT)'

New-ADUser @NewUserHT

1. Create an OU and move users into it:

$OUHT = @{

Name = 'IT'

DisplayName = 'Reskit IT Team'

Path = 'DC=Reskit,DC=Org'

}

New-ADOrganizationalUnit @OUHT # create the eOUI

$MHT1 = @{

Identity = 'CN=ThomasL,CN=Users,DC=Reskit,DC=ORG'

TargetPath = 'OU=IT,DC=Reskit,DC=Org'

}

Move-ADObject @MHT1 # move ThomasL into OU

$MHT2 = @{

Identity = 'CN=Rebecca Tanner,CN=Users,DC=Reskit,DC=ORG'

TargetPath = 'OU=IT,DC=Reskit,DC=Org'

}

Move-ADObject @MHT2 # Move Rebecca into OU

1. Create a third user directly in the IT OU:

$NewUserHT.SamAccountName = 'JerryG'

$NewUserHT.UserPrincipalName = 'jerryg@reskit.org'

$NewUserHT.Description = 'Virtualization Team'

$NewUserHT.Name = 'Jerry Garcia'

$NewUserHT.DisplayName = 'Jerry Garcia (IT)'

$NewUserHT.Path = 'OU=IT,DC=Reskit,DC=Org'

New-ADUser @NewUserHT

1. Add two users who then are to be removed:

# First user to be removed

$NewUserHT.SamAccountName = 'TBR1'

$NewUserHT.UserPrincipalName = 'tbr@reskit.org'

$NewUserHT.Name = 'TBR1'

$NewUserHT.DisplayName = 'User to be removed'

$NewUserHT.Path = 'OU=IT,DC=Reskit,DC=Org'

New-ADUser @NewUserHT

# Second user to be removed

$NewUserHT.SamAccountName = 'TBR2'

$NewUserHT.UserPrincipalName = 'tbr2@reskit.org'

$NewUserHT.Name = 'TBR2'

New-ADUser @NewUserHT

1. View the users that exist so far:

Get-ADUser -Filter \* -Property \*|

Format-Table -Property Name, Displayname, SamAccountName

1. Remove via a Get | Remove pattern:

Get-ADUser -Identity 'CN=TBR1,OU=IT,DC=Reskit,DC=Org' |

Remove-ADUser -Confirm:$false

1. Remove directly from the distinguished name:

$RUHT = @{

Identity = 'CN=TBR2,OU=IT,DC=Reskit,DC=Org'

Confirm = $false

}

Remove-ADUser @RUHT

1. Update then display the user details:

$TLHT =@{

Identity = 'ThomasL'

OfficePhone = '4416835420'

Office = 'Cookham HQ'

EmailAddress = 'ThomasL@Reskit.Org'

GivenName = 'Thomas'

Surname = 'Lee'

HomePage = 'Https://tfl09.blogspot.com'

}

Set-ADUser @TLHT

Get-ADUser -Identity ThomasL -Properties \* |

Format-Table -Property DisplayName,Name,Office,

OfficePhone,EmailAddress

1. Create a new group:

$NGHT = @{

Name = 'IT Team'

Path = 'OU=IT,DC=Reskit,DC=org'

Description = 'All members of the IT Team'

GroupScope = 'DomainLocal'

}

New-ADGroup @NGHT

1. Move all the users in the IT OU into this group:

$SB = 'OU=IT,DC=Reskit,DC=Org'

$ItUsers = Get-ADUser -Filter \* -SearchBase $SB

Add-ADGroupMember -Identity 'IT Team' -Members $ItUsers

1. Display the members:

Get-ADGroupMember -Identity 'IT Team' |

Format-Table SamAccountName, DistinguishedName

1. Add a computer to the AD:

$NCHT = @{

Name = 'Wolf'

DNSHostName = 'Wolf.Reskit.Org'

Description = 'One for Jerry'

Path = 'OU=IT,DC=Reskit,DC=Org'

OperatingSystemVersion = 'Windows Server 2019 Data Center'

}

New-ADComputer @NCHT

1. View the computers in the Reskit.Org domain:

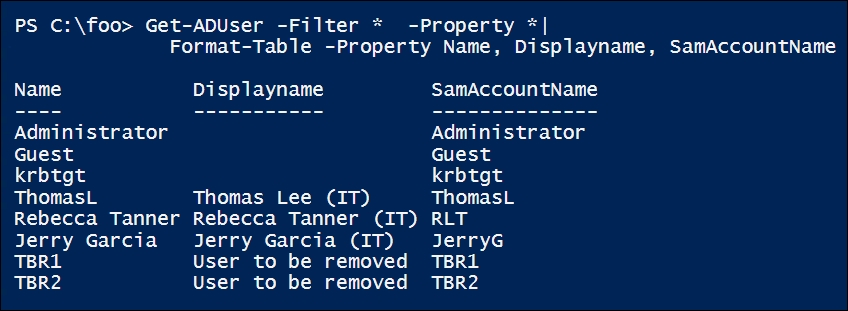
Get-ADComputer -Filter \* -Properties \* |

Format-Table Name, DNSHost\*,LastLogonDate

## How it works...

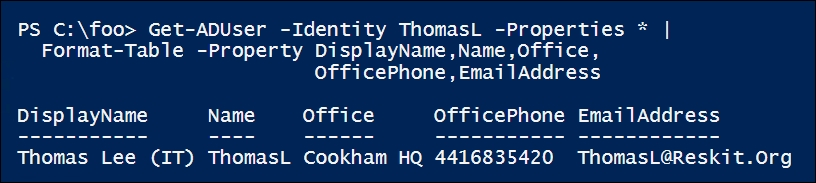
In step 1, you create a hash table of user properties to be set for the new user. You use this hash table in step 2 to create two users. In step 3, you create a new OU and move users into the new OU. In step 4, you add a third user directly into the OU. In step 5, you add two further users to the AD. These steps produce no output.

In step 6, you retrieve and display all the users in the Reskit.Org AD, like this:



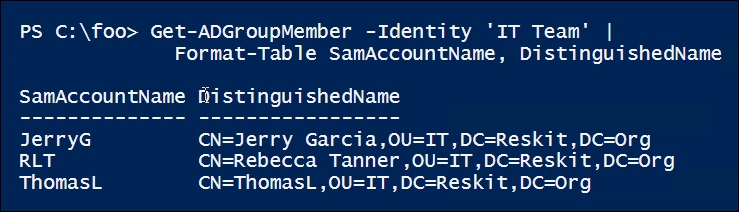
In step 7 and step 8, you remove two users using different removal patterns. The first user is removed via a Get | Remove pattern in which you get an object with a Get-ADUser cmdlet and then pipe it to the Remove-ADUser cmdlet. The second is the direct use of Remove-ADUser. Neither step produces any output.

In step 9, you update a user then display the updated user, which looks like this:

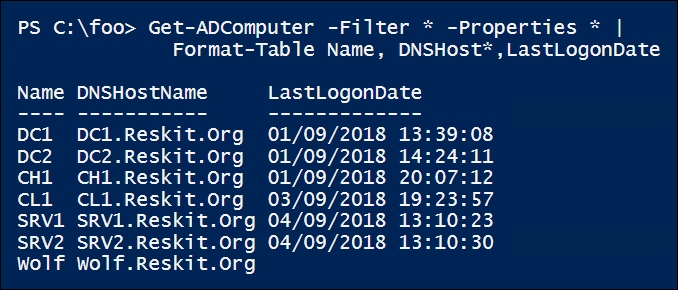


In step 10, you create a new domain local security group, IT Team, and in step 11, you populate the group membership with the users in the IT OU. Neither of these steps produces output.

In step 12, you display the users in the IT Team group, which looks like this:



In step 13, you pre-stage a computer account for a new host (Wolf.Reskit.Org), which produces no output. In step 14, you display all the computer accounts in the Reskit.Org domain, which looks like this:



## There's more...

In this recipe, we use two different approaches to creating a user. In the first, you use New-ADUser to create a new user which, by default, AD places in the User container in the domain. Since AD containers are not subject to a group policy, you need to move the created user objects into the appropriate OU in order to enable group policy and management delegation.

The second method used in this recipe to create a user involves using the PATH parameter to create the new user directly in an OU. When you create a new user, placing the user into the correct OU is better than leaving it in the Users container, especially if you want to apply GPOs to apply GPOs to the user.

The same logic applies to computer objects. By default, new computer objects are placed in the Computer container. As with users, objects in the Computer container are not subject to GPOs.

If you install Windows 2019 on a computer and promote it to be a domain controller, the domain installation process moves the computer account into the Domain Controllers OU. Computers in this OU are subject to a default GPO, the Default Domain Controllers policy.

You are also, in due course, likely to need to move users or computers between OUs. For example, the user Rebecca, created in step 2, might have initially been in the IT organization, but due to a job change, she moves to a new organization. To support that job move, you move her user and computer accounts to a different OU and change her group membership. After she reboots her computer and logs in, she acquires permissions needed for her new job and her user/computer accounts are then subject to any relevant GPOs.

In step 7 and step 8, you use two different methods for removing an object from the Active Directory (in this case, an AD user object). The first method is useful from the command line, where you first find the object(s) (using Get-ADUser) to be deleted, then pipe the results into Remove-ADUser.

The second way to remove an object is to use Remove-ADObject (or Remove-ADUser or Remove-ADComputer). Assuming you have the full distinguished name for the object to be removed, this is a bit faster. One risk of this approach is that you could accidentally type an incorrect DN into the command, resulting in the wrong user/computer being removed. To minimize this risk, consider configuring the AD recycle bin.

Joining a computer to the domain involves two specific steps: creating an account for the computer in the AD, then configuring the computer to be a member of the domain. If you build a new computer, you can log on as an administrator and join the computer to the domain, which achieves both of these steps. Alternatively, you could pre-stage a computer account (the first of the two necessary steps) then complete the domain-join process later.

In step 13, you pre-stage a computer account. This involves creating the AD account for a computer but without it actually joining the domain. This needs to be done using an account that has the necessary permissions. By default, this means a member of either the domain admins or the enterprise admins groups. After the computer account is created, you can complete the process of joining the computer to the domain (for example, by using the Add-Computer cmdlet). By pre-staging the account, a less privileged user can do the final step without the need for privileged domain credentials.

# Adding users to AD via a CSV file

As mentioned several times in this book, <https://www.spiceworks.com/> has a busy PowerShell support forum (accessible at <https://community.spiceworks.com/programming/powershell>). A frequently asked (and answered) question is: How do I add multiple users using an input file? This recipe does just that.

Start with a CSV file containing details of the users you are going to add. This recipe uses a CSV file and adds the users into the AD.

## Getting ready

This recipe assumes you have a domain setup and that you have created the IT OU. You did this in earlier recipes in this chapter. This recipe also requires a CSV file of users to add. You can create a CSV file like so:

$CSVDATA = @'

Firstname, Initials, LastName, UserPrincipalName, Alias, Description, Password

S,K,Masterly, skm, Sylvester, Data Team, Christmas42

C,B Smith, CBS, Claire, Claire, Receptionist, Christmas42

Billy-Bob, Joe-Bob, Bob, BBJB, BBJB, One of the Bob's, Christmas42

Malcolm, DoWrite, Duelittle, Malcolm, Malcolm, Mr Danger, Christmas42

'@

$CSVDATA | Out-File -FilePath C:\Foo\Users.Csv

## How to do it...

1. Import a CSV file containing the details of the users you wish to add to AD:

$Users = Import-CSV -Path C:\Foo\Users.Csv |

Sort-Object -Property Alias

$Users | Sort-Object -Property alias |FT

1. Add the users using the CSV:

ForEach ($User in $Users) {

# Create a hash table of properties to set on created user

$Prop = @{}

# Fill in values

$Prop.GivenName = $User.Firstname

$Prop.Initials = $User.Initials

$Prop.Surname = $User.Lastname

$Prop.UserPrincipalName = $User.UserPrincipalName+"@reskit.org"

$Prop.Displayname = $User.FirstName.trim() + " " +

$user.LastName.Trim()

$Prop.Description = $User.Description

$Prop.Name = $User.Alias

$PW = ConvertTo-SecureString -AsPlainText $user.password -Force

$Prop.AccountPassword = $PW

# To be safe!

$Prop.ChangePasswordAtLogon = $true

# Now create the user

New-ADUser @Prop -Path 'OU=IT,DC=Reskit,DC=ORG' -Enabled:$true

# Finally, display user created

"Created $($Prop.Displayname)"

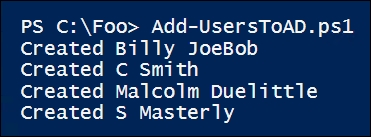
}

## How it works...

In step 1, you import the CSV file from C:\Foo\Users.Csv, which was noted in the Getting ready section of this recipe. Importing the CSV file generates no output.

In step 2, you iterate through the users in the CSV. For each user in the file, you first generate a hash table ($Prop) which you pass to the New-ADUser cmdlet to add the user to AD.

After you add each user, the recipe displays a message noting that the new user is now added to the AD. If you run the entire recipe as a single script, saved as Add-UsersToAD.ps1, and use the C:\foo\Users.Csv file created at the start of this recipe, then the output looks like this:



## There's more...

The basic approach of adding a user based on data in a CSV is straightforward. There are many variations on this approach that you can take depending on your circumstances.

You can expand the data included in the CSV file to populate more user properties for each AD user. For example, you could include a cell phone number, office address, and much more. Another variation is extending the CSV file and including one or more security groups that should have the user added.

# Creating a group policy object

Group policy allows you to define computer and user configuration settings that ensure a system is configured per policy.

With group policy, you first create a group policy object within the Active Directory. You then configure the GPO, for example, enabling computers in the IT organizational unit to be able to use PowerShell scripts on those systems. There are literally thousands of settings you can configure for a user or computer through group policy.

Once you configure your GPO object, you link the policy object to the OU you want to configure. You can also apply a GPO to the domain as a whole, to a specific AD site, or to any OU. A given GPO can be assigned in multiple places which can simplify your OU design.

The configuration of a GPO typically results in some data being generated that a host's group policy agent (the code that applies the GPO objects) can access. This information tells the agent how to work. Settings made through administrative templates use registry settings inside Registry.POL files. The group policy agent obtains the policy details from the SYSVOL share on a domain controller and applies them whenever a user logs on or off or when a computer starts up or shuts down.

The group policy module also provides the ability to create nice-looking reports describing the group policy object.

## Getting ready

This recipe runs on the DC1 domain controller that has been set up and configured in the three prior recipes in this chapter.

## How to do it...

1. Create a group policy object:

$Pol =

New-GPO -Name ITPolicy -Comment 'IT GPO' -Domain Reskit.Org

1. Ensure that only computer settings are enabled:

$Pol.GpoStatus = 'UserSettingsDisabled'

1. Configure the policy with two settings:

$EPHT1= @{

Name = 'ITPolicy'

Key = 'HKLM\Software\Policies\Microsoft\Windows\PowerShell'

ValueName = 'ExecutionPolicy'

Value = 'Unrestricted'

Type = 'String'

}

Set-GPRegistryValue @EPHT1 | Out-Null

$EPHT2= @{

Name = 'ITPolicy'

Key = 'HKLM\Software\Policies\Microsoft\Windows\PowerShell'

ValueName = 'EnableScripts'

Type = 'DWord'

Value = 1

}

Set-GPRegistryValue @EPHT2 | Out-Null

1. Create another GPO to disable the screen server, set the status, and add a description:

$Pol2 = New-GPO -Name 'Screen Saver Time Out'

$Pol2.GpoStatus = 'ComputerSettingsDisabled'

$Pol2.Description = '15 minute timeout'

1. Set a registry value:

$EPHT3= @{

Name = 'Screen Saver Time Out'

Key = 'HKCU\Software\Policies\Microsoft\Windows\'+

'Control Panel\Desktop'

ValueName = 'ScreenSaveTimeOut'

Value = 900

Type = 'DWord'

}

Set-GPRegistryValue @EPHT3 | Out-Null

1. Assign the GPOs to the IT OU:

$GPLHT1 = @{

Name = 'ITPolicy'

Target = 'OU=IT,DC=Reskit,DC=org'

}

New-GPLink @GPLHT1 | Out-Null

$GPLHT2 = @{

Name = 'Screen Saver Time Out'

Target = 'OU=IT,DC=Reskit,DC=org'

}

New-GPLink @GPLHT2 | Out-Null

1. Display the GPOs in the domain:

Get-GPO -All -Domain Reskit.Org |

Sort-Object -Property DisplayName |

Format-Table -Property Displayname, Description, GpoStatus

1. Create and view a GPO report:

$RPath = 'C:\Foo\GPOReport1.HTML'

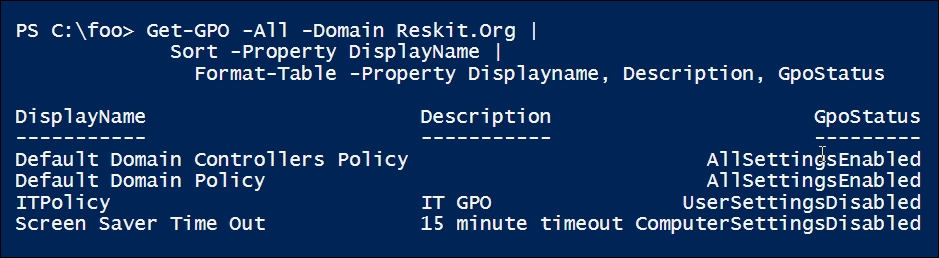
Get-GPOReport -Name 'ITPolicy' -ReportType Html -Path $RPath

Invoke-Item -Path $RPath

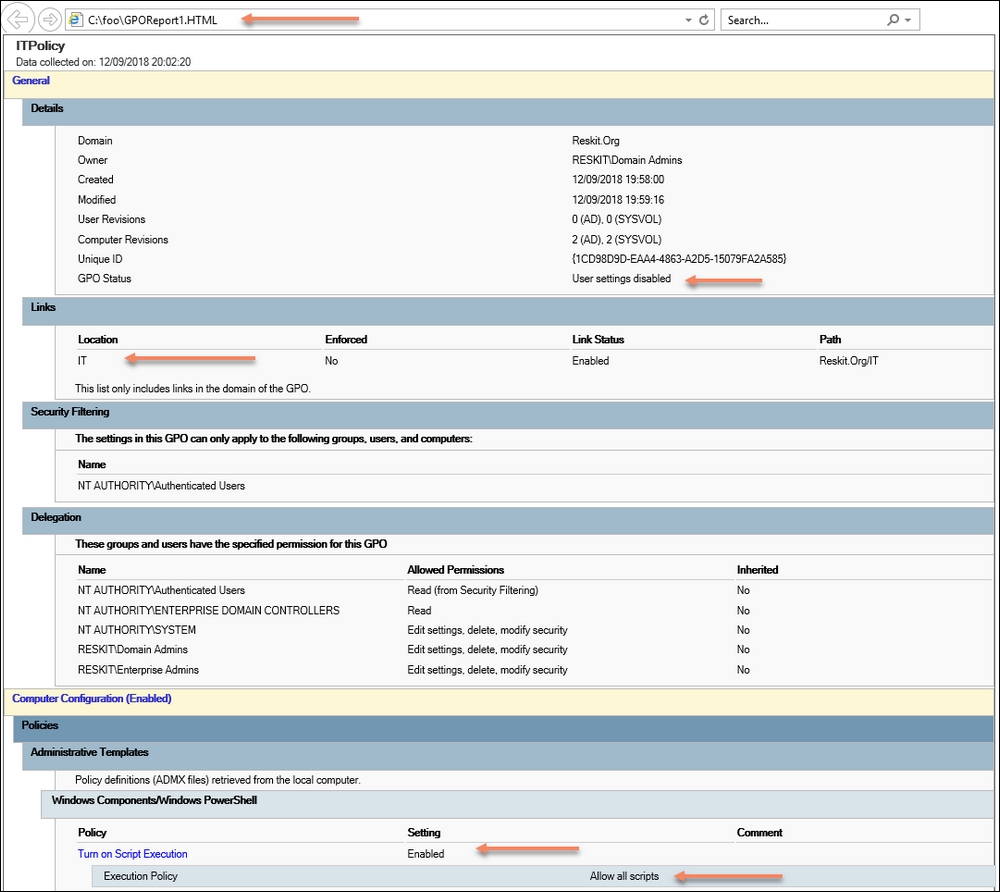
## How it works…

In step 1, you begin by creating a group policy object. In step 2, you update the GPO to indicate the GPO object has computer settings only. Next, in step 3, you configure the policy with two settings. In step 4 and step 5, you create a user GPO that sets values for the screen saver, which you apply in step 6. These steps produce no output.

In step 7, you display all the GPO objects in the Reskit.Org domain, which looks like this:



In step 8, you create and view a GPO report. This report shows the details of the GPO object, including the settings configured by the GPO, which looks like this:



## There's more...

Group policy is a rich topic—possibly one worthy of a book all of its own. In this recipe, you have seen the basics of creating, assigning, updating, and reporting on GPO objects. There is much more to group policy, including inheritance, backup/restore, export/import, and more.

## See also

With the group policy administrative templates, there are literally thousands of settings that are available—each with a registry setting for you to use, as shown in this recipe. Finding the specific settings and their respective registry values can be hard work. To assist you, Microsoft publishes an Excel spreadsheet that lists the settings. You can find this at <https://www.microsoft.com//download/details.aspx?displaylang=en&id=25250>.

# Reporting on AD users

Managing the Active Directory is an important albeit time-consuming task. Discovering a user account that has not been used for a reasonable period or a user that has membership in a privileged account (for example, enterprise administrators) could represent security risks to the organization. Regular reporting can help to place a focus on accounts that could be usefully de-activated. That could mean the account being removed from a security group or removed altogether.

This recipe creates a report of users, computers, and privileged group membership and displays this report on the console.

## Getting ready

This recipe, which you run on DC1, reports on users with possible issues: a user hasn't logged on for a while, has made a lot of bad password attempts, or a user is in a privileged group inappropriately.

## How to do it...

1. Define the Get-ReskitUser function:

Function Get-ReskitUser {

# Get PDC Emulator DC

$PrimaryDC = Get-ADDomainController -Discover -Service PrimaryDC

# Get Users

$ADUsers = Get-ADUser -Filter \* -Properties \* -Server $PrimaryDC

# Iterate through them and create $Userinfo hash table:

Foreach ($ADUser in $ADUsers) {

# Create a userinfo HT

$UserInfo = [Ordered] @{}

$UserInfo.SamAccountname = $ADUser.SamAccountName

$Userinfo.DisplayName = $ADUser.DisplayName

$UserInfo.Office = $ADUser.Office

$Userinfo.Enabled = $ADUser.Enabled

$userinfo.LastLogonDate = $ADUser.LastLogonDate

$UserInfo.ProfilePath = $ADUser.ProfilePath

$Userinfo.ScriptPath = $ADUser.ScriptPath

$UserInfo.BadPWDCount = $ADUser.badPwdCount

New-Object -TypeName PSObject -Property $UserInfo

}

} # end of function

1. Get the users in the Reskit.Org domain:

$RKUsers = Get-ReskitUser

# Build the report header:

$RKReport = ''

$RkReport += "\*\*\* Reskit.Org AD Report`n"

$RKReport += "\*\*\* Generated [$(Get-Date)]`n"

$RKReport += "\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*`n`n"

1. Report on the disabled users:

$RkReport += "\*\*\* Disabled Users`n"

$RKReport += $RKUsers |

Where-Object {$\_.Enabled -NE $true} |

Format-Table -Property SamAccountName, Displayname |

Out-String

1. Report on the users who have not recently logged on:

$OneWeekAgo = (Get-Date).AddDays(-7)

$RKReport += "`n\*\*\* Users Not logged in since $OneWeekAgo`n"

$RkReport += $RKUsers |

Where-Object {$\_.Enabled -and $\_.LastLogonDate -le $OneWeekAgo} |

Sort-Object -Property LastlogonDate |

Format-Table -Property SamAccountName,lastlogondate |

Out-String

1. Users with high invalid password attempts:

$RKReport += "`n\*\*\* High Number of Bad Password Attempts`n"

$RKReport += $RKUsers | Where-Object BadPwdCount -ge 5 |

Format-Table -Property SamAccountName, BadPwdCount |

Out-String

1. Add another report header line for this part of the report and create an empty array of privileged users:

$RKReport += "`n\*\*\* Privileged User Report`n"

$PUsers = @()

1. Query the enterprise admins/domain admins/schema admins groups for members and add to the $Pusers array:

# Get Enterprise Admins group members

$Members =

Get-ADGroupMember -Identity 'Enterprise Admins' -Recursive |

Sort-Object -Property Name

$PUsers += foreach ($Member in $Members) {

Get-ADUser -Identity $Member.SID -Properties \* |

Select-Object -Property Name,

@{Name='Group';expression={'Enterprise Admins'}},

whenCreated,LastlogonDate

}

# Get Domain Admins group members

$Members =

Get-ADGroupMember -Identity 'Domain Admins' -Recursive |

Sort-Object -Property Name

$PUsers += Foreach ($Member in $Members)

{Get-ADUser -Identity $member.SID -Properties \* |

Select-Object -Property Name,

@{Name='Group';expression={'Domain Admins'}},

WhenCreated, Lastlogondate,SamAccountName

}

# Get Schema Admins members

$Members =

Get-ADGroupMember -Identity 'Schema Admins' -Recursive |

Sort-Object Name

$PUsers += Foreach ($Member in $Members) {

Get-ADUser -Identity $member.SID -Properties \* |

Select-Object -Property Name,

@{Name='Group';expression={'Schema Admins'}},

WhenCreated, Lastlogondate,SamAccountName

}

1. Add the special users to the report:

$RKReport += $PUsers | Out-String

1. Display the report to the console:

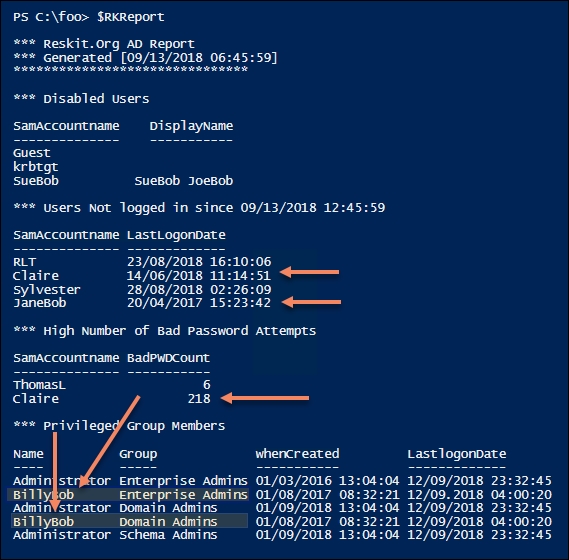
$RKReport

## How it works...

This report writing recipe begins with step 1 defining a function that returns some of the key properties of users in the Reskit.Org domain. In step 2, you invoke the function to return an array of all the users defined in the domain. These two steps produce no output.

In step 2 through step 9, you build parts of the overall report and add it to the $RKReport variable. These steps also produce no output.

In step 10, you display the report to the console, which looks like this:



## There's more...

In this recipe, you create the report and display it to the console. There are some things you could do that might increase the value of this recipe:

* Save the recipe as a script, then set up a scheduled task to invoke it every day
* Create an SMB share on the DC and save each report to that folder or email it to a mail-enabled group
* Automatically disable any account that has not been in use for, say, 30 days, sending mail to the user's manager letting them know
* Create a list of users authorized to be in the high-privilege groups and ensure the group contains only those members
* Adjust the recipe to output HTML to improve the readability and usability of the report

This recipe calls Get-ADUser several times in step 7, returning all properties. You might consider some optimization including restricting the properties to only those needed to generate the report.

# Finding expired computers and disabled users in AD

The objects in your AD database—the users, computers, groups, OUs, policies, and so on, are constantly changing in almost all organizations. Users leave, computers die, OUs and policies are added/removed/renamed, and so on. Change is constant!

A side effect of this change is having orphaned objects: users who are no longer part of your organization, or computers that no longer actually exist physically. You can also find you have objects that may be valid but have not been used for a long time.

Those accounts represent a potential security risk. An unused user account, for example, due to a user leaving and their account not being removed, can represent a threat vector. Suppose Ruth in the accounting department (who has access to the firm's accounting data) has left. If her account is active, then someone guessing her password could attempt to use her credentials to access such information. The risk is magnified if Ruth could access that information from the internet.

Any expired computers (that is, ones that have not logged in for a long time) may no longer have a machine password synced with AD). This means the computer is probably not getting WSUS updates or GPO-based policies.

This recipe finds computers that have not been used and users that have not logged in for a month. The recipe then generates a nice report and saves that report to a file in a corporate file share for you and others to look at.

## Getting ready

Run this recipe on DC1. Ideally, your domain should have enough users and computers created and configured.

## How to do it...

1. Build the report header:

$RKReport = ''

$RkReport += "\*\*\* Reskit.Org AD Daily AD report`n"

$RKReport += "\*\*\* Generated [$(Get-Date)]`n"

$RKReport += "\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*`n`n"

1. Report on the computer accounts that have not logged in the past month:

$RkReport += "\*\*\* Machines not logged on in past month`n"

$AMonthAgo = (Get-Date).AddMonths(-1)

$ADCHT2 = @{

Properties = 'lastLogonDate'

Filter = 'lastLogonDate -lt $AMonthAgo'

}

$RkReport += Get-ADComputer @ADCHT2 |

Sort-Object -Property lastLogonDate |

Format-Table -Property Name, LastLogonDate |

Out-String

1. Get the users who have not logged on in the past month:

$RKReport += "\*\*\* Users not logged on in past month`n"

$RkReport += Get-AdUser @ADCHT2 |

Sort-Object -Property lastLogonDate |

Format-Table -Property Name, LastLogonDate |

Out-String

1. Find any user accounts that are disabled:

$ADCHT3 = @{

Properties = 'Enabled'

}

$RKReport += "\*\*\* Disabled Users`n"

$RkReport += Get-ADUser @Adcht3 -Filter {Enabled -ne $true}|

Sort-Object -Property lastLogonDate |

Format-Table -Property Name, LastLogonDate |

Out-String

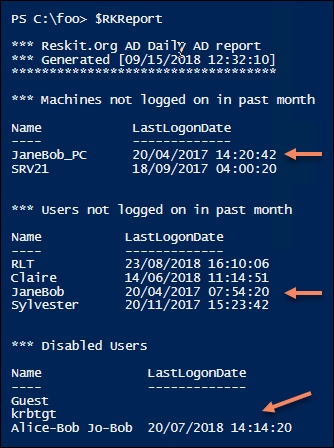
1. Display the report:

$RKReport

## How it works...

In step 1, you build a header for your report. In step 2, you add a list of computer accounts that have not signed on recently, and in step 3, you list the users who have not logged in for a while. In step 4, you add to the report details of disabled accounts. These first four steps produce no output.

In step 5, you display the report, which looks like this:



## There's more...

In step 5, you can see some things to consider in the report. One user has not logged on for a very long time (and her system hasn't either). You can also see users who are disabled. The first two (Guest and krbtgt) are normal and are to be expected. The final entry shows a user who is disabled and has not logged on for a very long time. Both user accounts should be reviewed to see if they are still needed by the business.