Active Directory with PowerShell

Windows PowerShell is a task-based command-line shell and is gaining popularity day-by-day. Using PowerShell to manage the Active Directory environment not only saves time for the system administrator, but end users also benefit as they see their requests being fulfilled in very little time.

The book starts with an overview of the components, software, and modules required to manage Active Directory with PowerShell. It then moves on to help you create and manage users, computer accounts, and group policies with simple examples to automate daily tasks. Furthermore, it covers topics such as GPOs, DNS Server/Client, DFS-N, and DFS-R automation. It also demonstrates how to automate some advanced operations so that they can be scripted to perform in a faster and more efficient way. By the end of this book, you will be competent enough to use PowerShell to manage your Active Directory environment and will gain all the required knowledge to automate your daily operations.

Who this book is written for
If you are looking to automate repetitive tasks in Active Directory management using the PowerShell module, then this book is for you. Any experience in PowerShell would be an added advantage.

What you will learn from this book
- Manage user and computer accounts using PowerShell
- Automate group membership additions, removals, and bulk operations using PowerShell
- Perform various query operations against Active Directory to fetch user, computer, and group details in an efficient and faster way
- Understand how sites, subnets, and domains are managed
- Perform advanced operations such as Domain Controller promotion/demotion
- Discover how to automate replication checks, fine-grained password policy creation, and FSMO roles transfer/seize using PowerShell
- Get to know more about DNS server management, record creation/modification/deletion, and DNS client management with PowerShell
- Find out ways to automate DFS-N and DFS-R installation and configuration using PowerShell

Learn to configure and manage Active Directory using PowerShell in an efficient and smart way.
In this package, you will find:

- The author biography
- A preview chapter from the book, Chapter 2 “Managing User and Computer Objects”
- A synopsis of the book’s content
- More information on Active Directory with PowerShell

About the Author

**Uma Yellapragada** has over 11 years of experience in the IT industry. Her core experience includes management of Active Directory, Microsoft Exchange, System Center Operations Manager (SCOM), Microsoft Office Communications Server (OCS/Lync), Microsoft Digital/Information Rights Management Services (DRMS/IRM), Hyper-V, VMware, PowerShell, and VBScript.

She also has experience working with process technologies such as ITIL, Six Sigma, and PMP.

She is the kind of person who challenges herself on a day-to-day basis and searches for areas of improvement as part of her work. As a result of this, she developed a passion for scripting with VBScript and PowerShell.

Active Directory with PowerShell

This book is for IT professionals who manage the Windows Active Directory infrastructure. Professionals supporting the Active Directory infrastructure, operations teams, and help desk members will find the content of this book useful. Any experience in PowerShell would be beneficial to help you easily grasp the content. Also, beginners can use this book to learn how to manage Active Directory environment using PowerShell.

What This Book Covers

Chapter 1, Let's Get Started, gives you an overview of the components, software, and modules required to manage Active Directory with PowerShell and gets you kick-started with routine tasks for automation. It also gives you the directions you need to use this book.

Chapter 2, Managing User and Computer Objects, helps users to perform various user and computer account administration related activities using PowerShell. By the end of this chapter, you will have a good understanding of how to manage user and computer Active Directory accounts using PowerShell and perform some of the automations based on it.

Chapter 3, Working with Active Directory Groups and Memberships, focuses on creating, modifying, and querying various kinds of security groups in Active Directory and their memberships. This chapter delivers the skills which are necessary for managing security groups in the Active Directory environment using PowerShell.

Chapter 4, Configuring Group Policies, helps in creating, linking, and unlinking Group Policies at various scopes; also, it is an integral part of Active Directory. By the end of this chapter, you will learn how to create GPOs, link them, enforce them, and perform several other operations using PowerShell. You will also be able to determine what policies are applied to a user and computer, remotely.

Chapter 5, Managing Domains, Organizational Units, Sites, and Subnets, tells you how to manage domains, Organizational Units, sites, and IP subnets using PowerShell. After completing this chapter, you will know how to manage OUs, sites, and IP subnets in your Active Directory environment.
Chapter 6, *Advanced AD Operations Using PowerShell*, talks about performing some of the advanced operations in Active directory such as promoting and demoting Active Directory domain controllers, the recovery of AD objects, and working with replication using PowerShell. After completing this chapter, you will know how to perform advanced AD operations, which are essential for any Active Directory administrator in a large enterprise environment.

Chapter 7, *Managing DFS-N and DFS-R Using PowerShell*, demonstrates how to create, configure, and query Distributed File System Namespace (DFS-N) and Distributed File System Replication (DFS-R) using PowerShell. By the end of this chapter, you will know how to administer DFS-N and DFS-R in a complex environment with the help of PowerShell.

Chapter 8, *Managing Active Directory DNS Using PowerShell*, helps you to understand how to manage AD DNS servers using PowerShell. A variety of operations such as clearing cache, creating and modifying records, working with zones, and many similar operations are covered in this chapter. By the end of this chapter, you will be able to manage Active Directory DNS servers using PowerShell to create, modify, and delete records, and perform some of the advanced DNS server operations.

Chapter 9, *Miscellaneous Scripts and Resources for Further Learning*, gives the information which you need about managing Active Directory using PowerShell. This will also provide references and code samples for some of the frequently performed Active Directory operations. By the end of this chapter, you will know where to look for further help.
Managing User and Computer Objects

In the previous chapter, we got familiarized with a list of tools required for managing Active Directory using PowerShell. Now, let's get started with actual management tasks and feel the real power of automation. This chapter enables you to learn automation of a few user and computer account-related operations.

This chapter mainly focuses on accomplishing the following tasks using PowerShell:

- Creating new user and computer accounts
- Modifying user and computer objects
- Enabling or disabling user and computer accounts
- Moving user and computer accounts
- Deleting user and computer accounts

Managing user accounts

Active Directory is all about users and computers. Each user in the organization will have at least one account. There will be scenarios where a single user can have multiple accounts. This is very true in the case of IT users where one account is used for regular activities such as checking emails, browsing, and so on, whereas, the other privileged account is used for managing the infrastructure. Apart from this, there are service accounts that are designed to run a particular service. This shows how rapidly user accounts can grow in the Active Directory environment along with the necessity to manage them in a much more efficient way.

The following sections will explain how to perform user object operations using PowerShell.
Managing User and Computer Objects

Creating user accounts
Managing user accounts is one of the day-to-day jobs as a Windows administrator. New users join companies on a frequent basis and sometimes the volume might go high. In such cases, creating user accounts using conventional methods is time-consuming and prone to errors. So, relying on automation for creating new users would be a wise choice and less time-consuming.

In Active Directory, the manual account creation process involves Graphical User Interface (GUI) tools, such as Active Directory Users and Computers (ADUC) or Active Directory Administrative Center (ADAC).

Let’s first take a look at how user creation can be done using ADAC.

ADAC was first introduced in Windows Server 2008 R2. It relies on Active Directory PowerShell cmdlets in Windows Server 2008 R2 and uses them in the background to perform the Active Directory operations. ADAC is further enhanced in Windows Server 2012 to expose the PowerShell commands that it uses in the background and repays study.

The AD module can be loaded into a normal PowerShell window using the following command:

```powershell
Import-Module ActiveDirectory
```

Ensure that you open your PowerShell window in the elevated mode (run as Administrator) to gain maximum benefits from the module.

Now, let's see the user creation process using a GUI tool named ADAC, shown in the following screenshot:
There are two mandatory fields that must be provided in order to create a user account: Full Name and User SamAccountName. Other fields are optional at the time of user creation and can be updated later. You might have also noticed that the password is not specified at the time of creation, so Active Directory keeps this field in a disabled state until the password is set. Once the password is set by the administrator, the user object has to be enabled explicitly.

Similarly, when a user account is created using PowerShell, it has one mandatory property that must be passed, the Name parameter. This parameter is equivalent to the Full Name value in UI. Also, the same parameter value is used for the user's SamAccountName attribute at the time of user account creation using PowerShell.

A user account in Active Directory can be created using the `New-ADUser` cmdlet. The following command is a small example to show how user account creation can be done:

```
New-ADUser -Name testuser1
```
Managing User and Computer Objects

When this command is executed from the PowerShell window, it creates a user account in the default user container. The account created will be in a disabled state because no password has been provided at the time of creation. This behavior is different when you create users using ADUC, where providing a password is mandatory.

The preceding one liner is just not sufficient for creating user accounts in the production environment. You are required to provide values for different attributes such as First Name, Last Name, Display Name, Password options (such as User must change password at next logon or not), Office address, phone numbers, Job title, Department, and the list goes on. So, we need to enhance our code to populate these properties at the time of login.

Before we start creating a full-fledged user account, let’s see which properties can be populated by the New-ADUser cmdlet at the time of user creation. You can get this simply by running the following help command:

    Get-Help New-ADUser -Detailed

The Get-Help cmdlet is the PowerShell cmdlet to see the help content of any other cmdlet. The usage of the -Detailed switch tells the Get-Help cmdlet to return all the help content for the given cmdlet. It includes a list of parameters, their syntax, an explanation of parameters, and examples.

To know more about each parameter, you can refer to the TechNet article at http://technet.microsoft.com/en-us/library/ee617253.aspx. This TechNet article explains about data types that each parameter stores and this is important to understand in order to read and write the attributes.

It is important to pay attention to the type of the value each parameter takes. If you provide any other data type apart from what it accepts, the New-ADUser cmdlet ends in an error. The type of information that each parameter takes can be identified from the TechNet page at http://technet.microsoft.com/en-us/library/ee617253.aspx.

As you can see in the preceding command, there are various properties (called attributes in AD terminology) that you can set at the time of user creation. If the attribute you want to set is not present, then you can use the OtherAttributes parameter to set it. Note that you need to provide other attribute names and values in hash table format while passing to the OtherAttributes parameter. Don’t worry about the use of hash tables. It is clearly explained later in this chapter in the Modifying user properties section.
Now, let's see how we can create a user account by passing all kinds of values that we want to set at the time of user creation. This example will cover some of the properties that are frequently used at the time of user object creation. However, you can modify this command and play around with setting other parameters. Practice makes one perfect!!!

The `Passthru` parameter is used to return the user object after creation of the account. If this parameter is not specified, the cmdlet will not show any output after successful creation of the object.

First, we need to prepare a password for the user to do the setting. Since the `-AccountPassword` cmdlet requires the input to be in secure string format, we need to populate the `$password` variable with the desired password, as shown by the following command:

```powershell
$Password = Read-Host "Enter the password that you want to set" -AsSecureString
```

This will prompt you to enter the password and you will see asterisk symbols as you enter. Ensure that the password you enter should meet the password complexity of your domain, otherwise the following command will fail:

```powershell
New-ADUser -Name Johnw -Surname "Williams" -GivenName "John" -EmailAddress "john.williams@techibee.ad" -SamAccountName "johnw" -AccountPassword $password -DisplayName "John Williams" -Department "Sales" -Country "US" -City "New York" -Path "OU=LAB,DC=Techibee,DC=AD" -Enabled $true -PassThru
```

Ensure that you update the `-Path` parameter in the preceding command to reflect the distinguished name of the OU in your environment. Otherwise, the operation might fail.

```
[ ] Note: The -Path parameter is optional. If you don't specify this, the user account will be created in the default users container.
```
Executing the preceding command from PowerShell will return the output shown in the following screenshot:

![PowerShell output screenshot]

The output shows the path of the object where it is created and other properties we set during the creation process. By default, the output shows only a minimum set of attributes. You can see all current attributes and the values of a user object using the `Get-ADUser` cmdlet:

```
Get-ADUser -Identity JohnW -Properties *
```

### Creating bulk user accounts

So far, we have seen how to create a single user account in Active Directory. Now, let's explore how to create multiple user accounts in one go.

The following command is sufficient enough if you want to create bulk user objects in the LAB environment without worrying about other properties such as department, email, and so on:

```
1..100 | foreach { New-ADUser -Name "Labuser$_" -AccountPassword $password -Path "OU=LAB,DC=techibee,DC=AD"}
```

It is a simple foreach loop that runs 100 times to create a user account with the name Labuser suffixed by the number of iteration (such as Labuser1, Labuser2, and so on.) with the password set to the value of the $password variable.

However, this is not sufficient to create user accounts in production environments. We need to populate several attributes at the time of creation. Ideally, system administrators receive the account creation information from HR in CSV format. So, the example being demonstrated in the following screenshot reads the account information from a CSV file, which has the details of user attributes, and creates user accounts based on this information.
Let's first read the contents of the CSV file into a PowerShell variable. A cmdlet called `Import-Csv` is available in PowerShell that can read the contents of the CSV file and return the output in object format. We can make use of it to read the contents, as shown in the following command:

```powershell
$Users = Import-Csv <path of the saved CSV file>
$Users | Format-Table
```

This command will read the contents of the CSV file into the `$Users` variable. The next statement will show the contents of the variable in table format. It should look as follows:

```
FirstName LastName Displayname Email LoginName Country City
Dave Williams Dave Williams Dave.Williams@techibee.com DaveW US New York
Chris Brown Chris Brown Chris.Brown@techibee.ad ChnsB UK London
John Miller John Miller John.Miller@techibee.ad JohnM US Redmond
```

Now, we have all user details in a variable, so let's proceed to create user accounts:

```powershell
foreach($User in $Users) {
    New-ADUser -Name $User.LoginName -Surname $User.LastName -GivenName $User.FirstName -EmailAddress $User.Email -SamAccountName $User.LoginName -AccountPassword $Password -DisplayName $User.DisplayName -Country $User.Country -City $User.City -Path "OU=LAB,DC=Techibee,DC=AD" -Enabled $true -PassThru
}
```

The preceding code will loop through each object in the `$Users` variable and invoke creation of user accounts by passing the properties of the `$User` object to parameters of the `New-ADUser` cmdlet. The value of the `-path` parameter in code has been hardcoded here, but you can make it part of the CSV file and pass it using the `-Path` parameter during creation. Executing this code will create three accounts in Active Directory with the details given in the CSV file.
All you need to do is populate the CSV file with the details you want to apply to each user object and prepare the `New-ADUser` cmdlet accordingly. Refer to Chapter 9, *Miscellaneous Scripts and Resources for Further Learning* for a more enhanced version of bulk user creation script.

### Modifying user properties

In the previous section, we have seen how to create user accounts using Active Directory PowerShell module. The task of the system administrator will not end by just creating user objects in Active Directory; he/she will also be responsible for modifying and managing them. This section will help you understand the process involved in modifying user accounts using PowerShell.

Since modifying user accounts has a very big scope, we will discuss a few example cases where a bulk user modification is required. These examples will help you understand the modification process. You can leverage these examples to modify any other attributes in Active Directory:

- Updating the description of a user object
- Updating the telephone number of multiple users
- Enabling or disabling user accounts in bulk
- Moving user accounts to another OU

Before jumping on to modifying user properties, let’s brush up on the basics basics. To update the description of a user account, you will typically follow these steps:

1. Open the ADAC tool (or ADUC).
2. Search for a username in Active Directory.
3. Go to the properties of the object.
4. Update the description and save your changes by clicking on **OK**.

What happens under the hood when you update the description of the object and save it? The system writes the value to a respective attribute of that user object. You can view all attributes and their values using the **Attribute Editor** tab in ADAC (or ADUC in Windows Server 2008 R2). To view this information from Windows Server 2003, you need to use the `adsiedit.msc` tool.

Here are the attribute details of user objects we created during bulk user creation in the previous section. You can see the values that are being read from the CSV file and used for creation.
So, in order to update any details of a user object, first we need to know its attribute name or display name.

Always remember that the names for properties you see in GUI tools might not be the same as what you see in the attribute editor. For example, the First Name field you see in GUI is translated to **givenName** in the attribute editor.

Similar to the GUI approach, we can search for a user object and list its attributes using PowerShell. The **Get-ADUser** cmdlet can be used for this:

```
Get-ADUser -Filter {Name -eq "ChrisB" }
```

This will return the user object with the Name attribute having the value **ChrisB**. By default, it will return only a basic set of attributes. If you want to see all attributes of this user object, then specify the `-Property` parameter, as shown in the following command:

```
Get-ADUser -Filter {Name -eq "ChrisB" } -Property *
```

You can also query users matching their name **Chris** by adjusting the value of the `-Filter` parameter, as shown in the following command:

```
Get-ADUser -Filter {Name -like "Chris*" }
```
Managing User and Computer Objects

Similarly, we can query all objects inside a particular Organizational Unit by passing the OU distinguished name to the -SearchBase parameter, as shown in the following command:

```
Get-ADUser -Filter * -SearchBase "OU=LAB,dc=techibee,dc=ad"
```

Now that we know how to search for objects, let's move on to learn how to modify them.

**Updating the description of a user object**

In the previous section, you learned how to search and find a user object. Let's use that logic and get an instance of the ChrisB user object, as shown in the following command:

```
$UserObj = Get-ADUser -Filter {Name -eq "ChrisB" } -Properties *
```

The $UserObj variable stores the reference to the ChrisB user object. We can view the current description of the user object by using the following command, assuming the user account has a description set:

```
$UserObj.Description
```

To replace the current description with new value, first we need to understand what data type this attribute will accept. As we mentioned before, we can find this out using the GetType() method. You can invoke this method, as shown in the following command:

```
$userObj.Description.GetType()
```

The output of this command shows that it accepts data of string, as shown in the following screenshot:

```
PS C:\> $userObj.Description.GetType()
IsPublic  IsSerial  Name  BaseType
--------  -------  ----  --------
True      True     String System.Object
PS C:\>
```

To update the description to a new value, we need to use the Set-ADUser cmdlet and pass the $UserObj to the -Identity parameter and the string that you want to set in the description field to the -Description parameter. The following command will return to the PS prompt in the PowerShell window if it completes without any errors:

```
Set-ADUser -Identity $UserObj -Description "Added new description via PowerShell"
```
To verify if the new description is updated in the user description field, we can either check it through the GUI or run the PowerShell command that we used for querying the user object in the preceding command.

Putting everything together, the following code will update the description of a given Active Directory user:

```powershell
$UserName = "ChrisB"
$NewDescription = "Delete this account after November"
$userObj = Get-ADUser -Filter {Name -eq $UserName} -Properties *
Write-Host "Current description is : $($userObj.Description)"
Set-ADUser -Identity $userObj -Description $NewDescription
$userObj = Get-ADUser -Filter {Name -eq $UserName} -Properties *
Write-Host "New description is : $($userObj.Description)"
```

You can update values of any other attribute using the procedure explained in this code. As a matter of practice, try updating the DisplayName of a user.

Remember? Practice makes perfect!!!

## Updating the telephone numbers of multiple users

In the previous example, you learned how to update the value of description using PowerShell. Now, let's take a look at updating telephone numbers of multiple users. This operation is a little different from updating the description operation. Here, we have two complexities, which are as follows:

1. We don't know which attribute will get updated when a number is added to the telephone number field in the GUI.
2. Performing a telephone number update for multiple users by reading from a text file or CSV file.
Let's address these complexities one by one. First, we need to identify attributes that need to be updated. In this demonstration, we want to update the **Office** telephone number, **Home** number, and **mobile** number of the users. If these users already have these numbers set, then we can use the attribute editor to identify the attribute names that need to be updated. If not, set these telephone numbers for one user account and then use attribute editor to identify the attribute names. Alternatively, you can use the `Get-ADUser` cmdlet to retrieve all attributes. The PowerShell way is preferred here, as we want to learn more and more about it. Examine the output of the following command carefully and identify the attribute names that have the telephone numbers you see in the GUI:

```powershell
Get-ADUser -Identity chrisB -Properties *
```

You will notice that the following command has the telephone numbers you see in the GUI:

```powershell
Get-ADUser -Identity ChrisB -Properties * | select HomePhone, OfficePhone, mobile
```

The description of the preceding command is shown in the following:

- **OfficePhone**: This attribute contains the phone number that you see under the **Main** field in telephone numbers in ADAC. In ADUC, you will see this number in the **Telephone number** field in the **General** tab.
- **HomePhone**: This is the phone number you will see in the **Home Phone** field.
- **mobile**: This attribute contains the phone number you will see in the **Mobile** field.

First, let's update the telephone details of one user, then we can extend the logic to update other users in bulk.

Let's store the numbers in variables first, as shown in the following commands:

```powershell
$OfficeNumber = "+65 12345678"
$HomeNumber = "+65 87654321"
$MobileNumber = "+65 13578642"
```

The next step is to update the preceding values for a user account, as shown in the following command:

```powershell
Set-ADUser -Identity ChrisB -OfficePhone $OfficeNumber -HomePhone $HomeNumber -MobilePhone $MobileNumber
```
This is straightforward because the `Set-ADUser` cmdlet has parameters that can set these phone numbers. If the attribute that you are trying to set is not available as a parameter to the cmdlet then you can use the `-Add` parameter to directly specify the attribute name and the value. Similarly, you can use other parameters such as `-Replace` and `-Clear` to work with attributes directly. The preceding example can be rewritten using the `-Add` parameter, as shown in the following commands:

```powershell
Set-ADUser -Identity ChrisB -Clear telephonenumber, homephone, mobile
Set-ADUser -Identity ChrisB -Add @{telephonenumber = $OfficeNumber; homephone = $HomeNumber; mobile = $MobileNumber}
```

Here, we are setting the office phone number to the `telephonenumber` attribute, the home number to the `homephone` attribute, and the mobile number to the `mobile` attribute. Before setting them, we will clear existing values using the `-Clear` parameter.

Now, we can extend this logic to multiple users using a for loop in PowerShell. Before doing this, store the user names and numbers you want to set in a CSV file and import it into PowerShell. The following screenshot shows how the contents of the CSV look:

<table>
<thead>
<tr>
<th></th>
<th>A</th>
<th>B</th>
<th>C</th>
<th>D</th>
</tr>
</thead>
<tbody>
<tr>
<td>1</td>
<td>UserName</td>
<td>OfficeNumber</td>
<td>HomeNumber</td>
<td>MobileNumber</td>
</tr>
<tr>
<td>2</td>
<td>ChrisB</td>
<td>+65 12345678</td>
<td>+65 87654321</td>
<td>+65 13578642</td>
</tr>
<tr>
<td>3</td>
<td>SteveR</td>
<td>+65 12345678</td>
<td>+65 97054367</td>
<td>+65 43561209</td>
</tr>
</tbody>
</table>

And the code to set the telephone numbers is as follows:

```powershell
$Users = Import-Csv c:\temp\usersPhoneNumbers.csv
foreach($User in $Users) {
}
```

### Enabling or disabling user accounts

Now, let's take a look at another scenario where we want to perform bulk user enable/disable operation. Fortunately, there are two cmdlets provided in Active Directory module to make this operation very easy and straightforward.
Managing User and Computer Objects

They are as follows:

- **Enable-ADAccount**: This cmdlet is used for enabling Active Directory user, computer, or service account objects
- **Disable-ADAccount**: This cmdlet is used for disabling Active Directory user, computer, or service account objects

Both these cmdlets require an object to be enabled/disabled. The object can be in one of the following formats:

- Distinguished Name (DN) format, for example, `CN=ChrisB, OU=LAB, DC=techibee, and DC=ad`
- ObjectGUID format, for example, `923199de-0dd9-4758-a954-5aa42409b10d`
- Security Identifier (SID) format, for example `S-1-5-21-822638036-2026389545-1116158610-1244`
- **SAMAccountName** format, for example, `ChrisB`

To get these values for a given user, use the **Get-ADUser** cmdlet (use **Get-ADComputer** in the case of computer accounts)

Now, it's just a matter of passing the input values to either the **Enable-ADAccount** or **Disable-ADAccount** cmdlets based on which operation you want to perform.

Here are some common usage scenarios. These scenarios cover disable operations; to perform enable operations in a similar way, just replace the **Disable-ADAccount** cmdlet with the **Enable-ADAccount** cmdlet.

The following command can be used to disable a single user account:

```
Disable-ADAccount -Identity ChrisB -Passthru
```

The **-PassThru** parameter is used to return the object after the completion of the operation. Also it is useful to know the disable status if you want to perform further actions on this object.

You can disable users in a particular OU. The following command will return all users objects under LAB OU and its sub OUs:

```
Get-ADUser -SearchBase "OU=LAB,DC=techibee,DC=AD" -Filter * | Disable-ADAccount
```

To limit the search scope to the current OU, use the **-SearchScope** parameter. It takes three values: Basic (or 0), OneLevel (or 1), a Subtree (or 2). Subtree is the default value when nothing is specified.
Read usernames from a text file and disable them, as shown in the following command:

```powershell
Get-Content C:\temp\users.txt | % { Disable-ADAccount -Identity $_ }```

Here, the `Get-Content` cmdlet reads the usernames from the text file and passes them one by one to the `Disable-ADAccount` cmdlet using a foreach loop ($_ is an alias for a foreach loop in PowerShell). When passing the user name to the `Disable-ADAccount` cmdlet, we use the $_ automatic variable, which contains the name that is passed from the pipeline. Read more about automatic variables at http://technet.microsoft.com/en-us/library/hh847768.aspx.

You can disable all users in a department, for example, Sales. The following command queries all users who have their department value set to sales and passes them to the `Disable-ADAccount` cmdlet to disable them:

```powershell
Get-ADUser -Filter 'Department -eq "sales"' | Disable-ADAccount```

Likewise, to perform an enable operation, just replace `Disable-ADAccount` with the `Enable-ADAccount` cmdlet in the preceding examples.

### Moving user accounts to another OU

For example, let's consider a scenario where all users of one department are moved from one office building to another. So, you would like to move all these user accounts to a new OU for ease of identification and management.

The `Move-ADObject` cmdlet is available in Active Directory module to accomplish this operation. As you might have already noted, this particular cmdlet can move any object from one OU to another; not just user accounts.

This cmdlet has two mandatory parameters:

- **Identity**: This identifies the object that you want to move. It can be either the Distinguished name (DN) of the object or the GUID of the object.
- **TargetPath**: The TargetPath parameter must be the Distinguished Name (DN) of OU or the container to which you want to move the objects.

Here are some use case scenarios:

- Moving a user account from one OU to another:

  ```powershell
  Move-ADObject -Identity "CN=ChrisB,OU=LAB,DC=techibee,DC=ad" -TargetPath "OU=Singapore,OU=LAB,DC=Techibee,DC=ad"
  ```

  Here, ChrisB is the name of the user that we are moving from the current location (specified in the DN) to the new -TargetPath parameter.
Managing User and Computer Objects

- Moving all users from LAB OU to PROD OU

```powershell
Get-ADUser -Filter * -SearchBase "OU=LAB,DC=techibee,DC=ad" | Move-ADObject -TargetPath "OU=Prod,DC=techibee,DC=ad"
```

The preceding command will move all users (including users in sub OUs) from LAB OU to PROD OU. The `-Identity` parameter is automatically populated from the output of the `Get-ADUser` cmdlet.

You can use the following command to move users from one OU to another based on their department name:

```powershell
Get-ADUser -Filter 'department -eq "Sales"' | Move-ADObject -TargetPath "OU=Sales,OU=PROD,DC=techibee,DC=AD"
```

Remember: The target OU must exist before you move users to it. The distinguished name of an object should always be unique; the common name portion of the distinguished name can, however, be used more than once.

### Deleting user accounts

The Active Directory PowerShell module has a cmdlet called `Remove-ADUser` to delete user accounts from Active Directory. Alternatively, the `Remove-ADObject` cmdlet can be used. The `Remove-ADUser` cmdlet is designed to deal with user accounts removal. We will use this cmdlet throughout the examples in this section.

The requests for removal of user accounts increase as the attrition rate increases in your organization. You get requests from HR to delete user accounts on a frequent basis either when an employee leaves the organization or he/she turns down the offer just before joining.

Most organizations won't delete user accounts when an employee leaves the organization. Instead, they will hide these from the Global Address List/Book (GAL), remove them from all groups, disable the mailbox, and keep the ID in a disabled state. Such accounts can be enabled if the employee rejoins the company later.

Removing a user account from Active Directory is a straightforward process. You just need to pass the DN or ObjectGUID, SID or SamAccountName to the `-Identity` parameter of the `Remove-ADUser` cmdlet. In the following example, `Samaccountname` is passed to the `-Identity` parameter, as shown in the following command:

```powershell
Remove-ADUser -Identity ChrisB
```
When this command is executed, it will ask for confirmation of deletion, as shown in the following screenshot:

Since deletion is a critical operation, Active Directory module warns about it. If you are certain that the inputs are correct and you don't want to get prompted for confirmation, set the `-Confirm` parameter value to `$false`, as shown in the following command:

```
Remove-ADUser -Identity ChrisB -Confirm:$false
```

Similarly, to delete user accounts by reading from a text file, use the following command:

```
Get-Content C:\temp\users.txt | % { Remove-ADUser -Identity $_ -Confirm:$false}
```

The `Get-Content` cmdlet reads the usernames from `users.txt` and passes them to the `Remove-ADUser` cmdlet to delete the accounts one after another.

**Managing computer accounts**

In previous sections, we have seen several operational tasks that can be performed on user accounts in Active Directory using PowerShell. This section focuses on performing similar operations on computer objects.

Managing computer objects is not much different from managing user objects. All you need to do is to use the correct cmdlets and the rest of the process remains the same.

The following topics are covered as a part of managing computer accounts using PowerShell. Let's go through these one by one and understand how we can accomplish them:

- Creating computer accounts
- Modifying computer properties
- Enabling or disabling computer accounts
- Deleting computer accounts
Creating computer accounts

Most system administrators do not create computer accounts manually in Active Directory. Instead, they join computers to the domain and the account gets created automatically. After automatic object creation, the administrator moves the computer accounts from the default container to the desired OU.

Well, this might look quite simple but why is there a need to create a manual computer account? The aforementioned approach will work for small organizations where one set of system administrators will manage everything and they will have all privileges. However, in large organizations, this is not feasible for various reasons. In large organizations the desktop/server builds happen in an automated way using deployment solutions, such as Windows Development Services (WDS) where the build process looks for a computer account in AD to join the server/desktop to the domain. This process is called prestaging of computer accounts and it has a good set of advantages such as choosing the OU where you want to place the computer, group membership, and so on.

So, let's now look at a few examples of creating a computer account. Active Directory provides a cmdlet called *New-ADComputer* to facilitate the computer account creation.

The following command will create a computer account with the name **SRVMEM2** in the default computers container:

```
New-ADComputer -Name SRVMEM2 -PassThru
```

To create computer account in a particular Organizational Unit in Active Directory, use the following command:

```
New-ADComputer -Name SRVMEM2 -Path
 "OU=Computers,OU=PROD,DC=techibee,DC=AD" -PassThru
```

Ensure that the OU mentioned in the preceding command exists prior to the computer account creation; if not, the command execution fails.

If you just want to create the computer account but keep it in a disabled state, the following command helps:

```
New-ADComputer -Name SRVMEM2 -Path
 "OU=Computers,OU=PROD,DC=techibee,DC=AD" -Enabled $false -PassThru
```

Notice: The `-Enable` parameter in the preceding command, which is set to `$false`, is responsible for disabling the computer account.
To see the list of other options for this cmdlet and some examples, read its complete help content. This can done using the following command:

```
Get-Help New-ADComputer -Full
```

### Modifying computer accounts

Computer account attributes often need to be modified. For example, because many computer accounts are created before the computers are actually assigned to users, attributes such as description, department, and location cannot be configured at the time an account is created. In addition, the ownership of a computer can be transferred to a new user or department, or a computer might be physically moved to a new location. In such circumstances, the computer account attributes need to be modified.

Let's see the PowerShell way of doing this using various cmdlets available for computer objects.

### Setting the description for a computer account

Active Directory PowerShell module has the `Set-ADComputer` cmdlet for modifying computer account properties in Active Directory. Remember the `Set-ADUser` cmdlet we used for modifying user object properties? It is similar to that but for computer accounts.

To update the description of a single computer, you can use the following command. This example updates the description of the `SRVMEM1` computer object:

```
Set-ADComputer -identity SRVMEM1 -description "Member Server"
```

The `Set-ADComputer` cmdlet has ability to set values for the majority of object attributes. To see the list of attributes it can set, check its help content using the following command:

```
Get-Help Set-ADComputer -Full
```
Managing User and Computer Objects

You can use the `Get-ADComputer` command to check if the description is set as shown in the following screenshot:

![PowerShell screenshot](image.png)

**Moving computer accounts to a different OU**

Sometimes, you might need to move computer accounts to different OUs as the user might change the location or server accounts in order to segregate them according to their roles; or you might want to move computer accounts from the default OU to respective office location OUs.

Let's see a few of the examples related to computer account movements across Organizational Units. As we have done for user accounts, here too we can make use of the `Move-ADObject` cmdlet to move computer accounts from one OU to another. The following command moves the `SRVMEM1` computer account from the default computer container to Computers OU inside PROD OU:

```
Move-ADObject -Identity "CN=SRVMEM1,CN=Computers,DC=techibee,DC=ad" -TargetPath "OU=Computers,OU=PROD,DC=techibee,DC=ad" -PassThru
```

Since it is not possible to provide the full DN of the object we want to move every time, we can either use the `Get-ADComputer` or `Search-ADAccount` cmdlet to search by its name or some other property and then pass the output to the `Move-ADObject` cmdlet. The following example demonstrates this:

```
Get-ADComputer -Filter "name -eq 'SRVMEM1'" | Move-ADObject -TargetPath "OU=Computers,OU=PROD,DC=techibee,DC=ad" -PassThru
```

Similarly, we search for a string in the description of the computer objects and move them to the designated OU using the following command:

```
Get-ADComputer -Filter "description -like '*server*'" | Move-ADObject -TargetPath "OU=Computers,OU=PROD,DC=techibee,DC=ad" -PassThru
```
This command will look for computer accounts that have the string server in their description and will move them to the designated OU. Similarly, you can search based on any other criteria and move them to different OUs, as shown in the preceding command.

**Enabling or disabling computer accounts**

As a system administrator, it is required to keep your Active Directory database clean, tidy, and minimal in size. Also, one must adhere to the security policies of the organization and often needs to reconcile computer accounts data monthly, quarterly, and annually in accordance with those security policies.

The cmdlet we use here is not specific to computer objects; it can also be used for any Active Directory user, computer, or service accounts.

Use the following command to enable a particular computer:

```
Get-ADComputer -Identity COMP1 | Enable-ADAccount
```

To enable multiple computer accounts, you can use filters in conjunction with the Get-ADComputer or Search-ADAccount cmdlets. The following command will search for computer accounts inside the given OU and it will enable all of them:

```
Get-ADComputer -Filter "*" -SearchBase
  "OU=Computers,OU=PROD,DC=techibee,DC=ad" | Enable-ADAccount -PassThru
```

Similarly, to disable computer accounts, just replace Enable-ADAccount with the Disable-ADAccount cmdlet. The following command disables all of the computers inside the given OU:

```
Get-ADComputer -Filter "*" -SearchBase
  "OU=Computers,OU=PROD,DC=techibee,DC=ad" | Disable-ADAccount -PassThru
```

Explore more by referring to the help content of these cmdlets. For example, to see the help of the Get-ADComputer cmdlet, you can use the following command:

```
Get-Help Get-ADComputer -Detailed
```

You can read the help content from TechNet site as well (http://technet.microsoft.com/en-us/library/ee617192.aspx).
Deleting computer accounts

As discussed in the previous section, as a system administrator one must adhere to the security policies of the organization and keep their Active Directory database clean and tidy. As part of this process, you might want to delete stale/offline computer objects from Active Directory.

Use the following simple command to delete a computer account:

```
Remove-ADComputer -Identity COMP1
```

The most common use case is searching for computers older than \( x \) days and removing them. You can achieve this using the following command:

```
$Computers = Get-ADComputer -Filter * -Properties LastLogonDate | ?
    {$_ . LastLogonDate -lt (get-date).AddDays(-10) }
$Computers | Remove-ADComputer
```

You need to be very careful while performing the delete operation. Any mistake in the filters can result in your production computers being deleted. So, I prefer storing the `Get-ADComputer` cmdlet results in a variable (`$computer` in this example), reviewing the list, and then passing it to the `Remove-ADComputer` cmdlet.

The first line in the preceding code searches Active Directory for computers that are not contacted in the last 30 days and stores them in a variable. Later, we can pass the variable to the `Remove-ADComputer` cmdlet to delete them. By default, this cmdlet will prompt for each deletion; you can override it using the `-Confirm:$false` property with the `Remove-ADComputer` cmdlet.

To delete multiple computer accounts that have location value set to `OFFICE1`, you can use the following command:

```
Get-ADComputer -filter 'Location -eq "OFFICE1"' | Remove-ADComputer -confirm:$false
```

Use the following command to delete all computer accounts in a particular OU:

```
Get-ADComputer -SearchBase "OU=DisabledComp,DC=techibee,DC=ad" | Remove-ADComputer -confirm:$false
```
These examples will help you to get started. For instance, you can use the `Get-ADComputer` cmdlet to search computer accounts using different patterns and pass them to the `Remove-ADComputer` cmdlet to delete them.

Deletion of user accounts or computer accounts is a critical operation and reverting these changes is not possible in all cases. So, you should verify what you are searching and removing. Review the output of the `Get-ADComputer` cmdlet twice before you pass it to the `Remove-ADComputer` cmdlet.

**Summary**

In this chapter, we have seen how to manage Active Directory users and computer accounts using PowerShell. The examples quoted in each section will help you to get started. You can enhance your learning further by reviewing the help content of each cmdlet that we have used throughout this chapter.

You might wonder why managing group membership of users and computers is not covered in this chapter. Considering the fact that group membership changes are more frequent in Active Directory environment, the next chapter focuses in depth on this topic. Various operations related to groups and their memberships are covered there in great detail.
Where to buy this book
You can buy Active Directory with PowerShell from the Packt Publishing website.
Alternatively, you can buy the book from Amazon, BN.com, Computer Manuals and most internet book retailers.
Click here for ordering and shipping details.