Conversational Scripted AD

A ConversationalGeek™ Book

Learn about:

- What's possible with Active Directory automation through scripting
- How to save time with PowerShell automation and Active Directory
- How to manage users, group and computers with PowerShell

By Adam Bertram Microsoft Windows PowerShell MVP

Conversational Scripted AD

By Adam Bertram

ConversationalGeek

Conversational Scripted AD

Published by Conversational Geek Inc.

www.conversationalgeek.com

Trademarks

Warning and Disclaimer

Additional Information

For general information on our other products and services, or how to create a custom Conversational Geek book for your business or organization, please visit our website at ConversationalGeek.com

Publisher Acknowledgments

All of the folks responsible for the creation of this guide:

Author:	Adam Bertram
Project Editor:	J. Peter Bruzzese
Copy Editor:	John Rugh
Content Reviewer(s):	Shanna G. Giarranno

Note from the Author

Welcome to Conversational Scripted AD! I'm Adam Bertram, a long time tech author and I am going to be giving you a quick crash course on the subject of scripting for Active Directory.

As you have probably guessed from the thickness of this book (or the lack thereof), this text isn't meant to be an exhaustive, super deep, hard core technical reference. Rather, it's me taking the time to explain some of the more important concepts in an effort to familiarize you with the subject. I'm not going to talk about every detail and possible command, but I am going to give you an introduction to the more important ones and spend some time talking about important concepts.

One last thing that I want to be sure to mention is that this book isn't intended to be a sales pitch. My goal is to teach you the basics of scripting with AD. With that said, enjoy the book!

Adam Bertram

The "Conversational" Method

We have two objectives when we create a "Conversational" book: First, to make sure it's written in a conversational tone so that it's fun and easy to read. Second, to make sure you, the reader, can immediately take what you read and include it into your own conversations (personal or business-focused) with confidence.

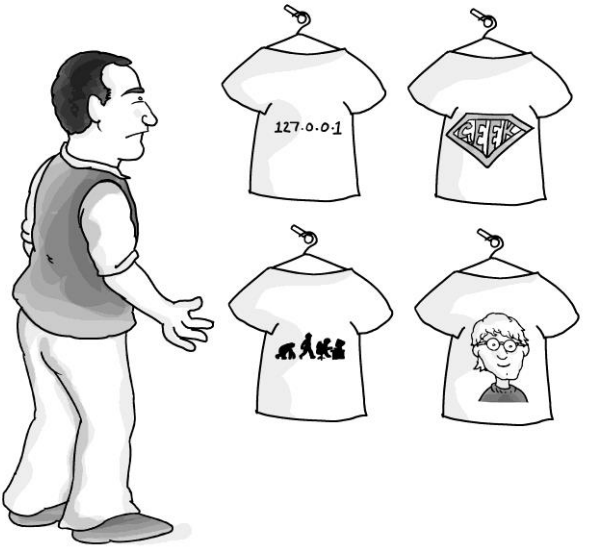
These books are meant to increase your understanding of the subject. Terminology, conceptual ideas, trends in the market, and even fringe subject matter are brought together to ensure you can engage your customer, team, co-worker, friend and even the know-it-all Best Buy geek on a level playing field.

"Geek in the Mirror" Boxes

We infuse humor into our books through both cartoons and light banter from the author. When you see one of these boxes it's the author stepping outside the dialog to speak directly to you. It might be an anecdote, it might be a personal experience or gut reaction and analysis, it might just be a sarcastic quip, but these "geek in the mirror" boxes are not to be skipped.

Greetings. I'm Adam's 'geek in the mirror' and within these boxes I can share just about anything on the subject at hand. Read 'em!

Active Directory Scripting with PowerShell



If you're working in an organization of just about any size you probably have some kind of Active Directory (AD) implementation in place. AD is a widely used product that many IT pros have quite a bit of experience with. Maybe you're in the helpdesk where you're creating new users or resetting passwords. Perhaps you're an AD administrator setting up new child domains, ensuring all the domain trusts are in place and ensuring the AD schema is solid. In any case, you've probably got some experience tinkering with it.

GUI Lover

Tools like Active Directory Users and Computers (ADUC) or Active Directory Administrative Center (ADAC) have probably been your go-to tools for a long time. Am I right? You might have gotten crazy and peeked into AD Sites and Services (ADSS) or maybe even got a wild hair and fired up ADSI Edit to make some schema changes. What do all of these tools have in common? No, they all don't start with an "A", smarta**. Well, they do, but...you get my point! They all have a GUI!

"What does that have to do with anything?", you may be asking. Maybe you're thinking, "So what? They're all GUI tools. There's nothing wrong with GUI tools. You gotta problem with GUI tools? Don't tell me you're one of those greasy long-haired command line geeks. Are you?" Well, I don't have long hair but if I don't wash my hair for a few days it might get a little shiny but that's beside the point. The point I'm trying to make here is that managing AD doesn't have to be done with any of these tools. You could manage AD all day and do everything you can from the GUI but never touch the mouse. If you get good enough, you'd be able to smoke your co-workers in AD tasks. But why would you want to do that?

Why Script Active Directory?

Before we get too far, let's take a minute and get a little more specific when I talk about "command line AD". This could mean a lot of things but I'm talking about using Windows PowerShell with AD. If you've never touched PowerShell before now you've been living under a rock or perhaps you're one of those "automation will take my job" kinda IT pros. Frankly, I'm surprised you've been able to stay employed this long! Learn PowerShell! Stepping down from my soapbox, seriously, if you're brand new to PowerShell I might be going over your head a little here. I highly suggest you check out the Conversational PowerShell eBook before you get much farther.

It will go over all the basics with you to get you up to speed in no time.

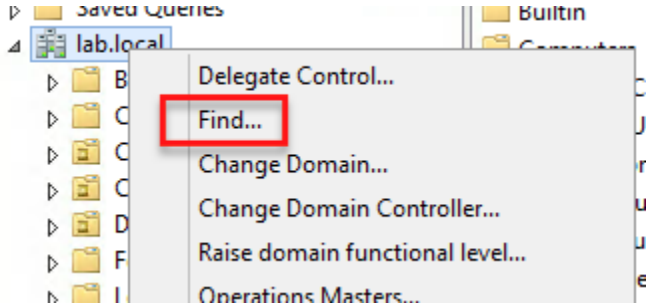
With that being said, let's go over the two reasons why you might want to script Active Directory with PowerShell.

For Quick, Ad-Hoc Tasks

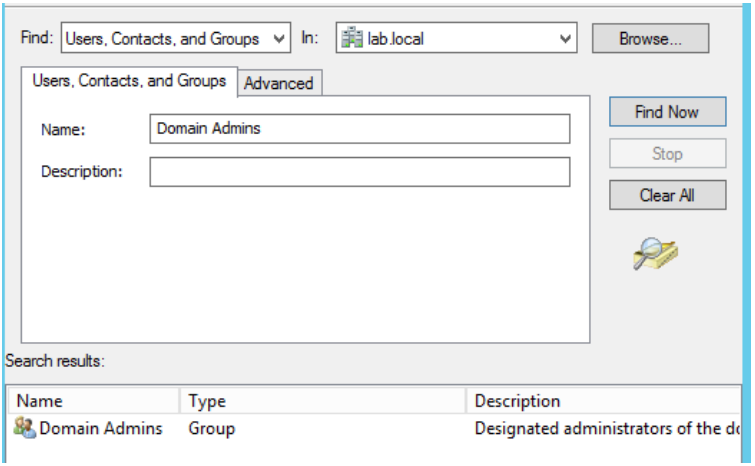
You probably do little, one-off tasks all the time like adding a user account, resetting passwords, moving a computer into another OU and so on. Some of these tasks are actually faster in the GUI. There I said it. But, lots of tasks are much faster whenever you try them with PowerShell. Let me give you an example.

The GUI

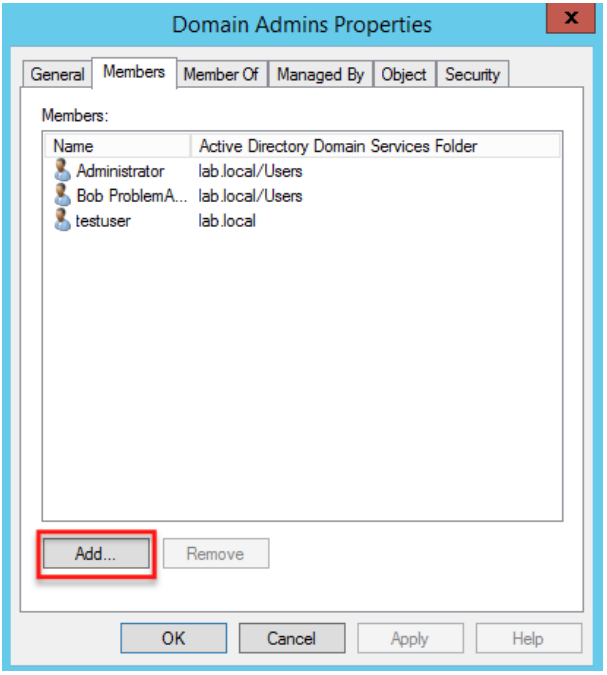
Let's say you need to add a user to the Domain Admins group. This is a simple task but look what's required if you use the GUI. First, you might right-click on your domain and click on *Find*.



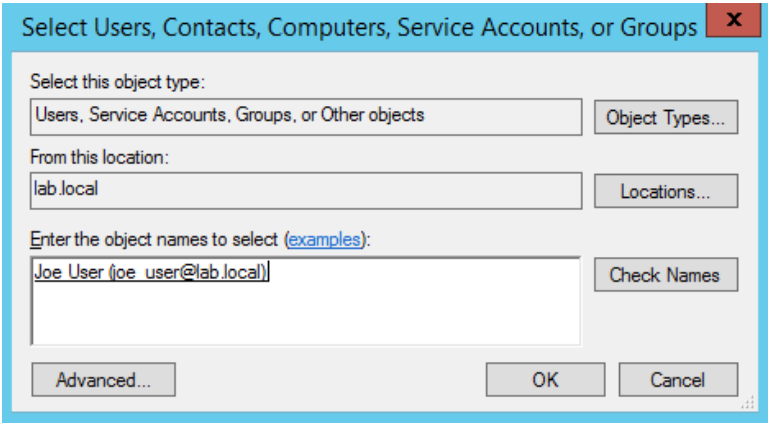
Next, you'd type in *Domain Admins*, hit Enter and then the group will show up in the bottom pane.



You'd then double click on Domain Admins which would bring up the familiar properties dialog box. At that point, you'd click on Add.



Then, at that point, another dialog box would come up where you'd type the user account you'd like to add and click on Check Names like we all do to ensure it's a valid account.



You'd then click OK to close the selection dialog box and then OK again on the Domain Admins properties box and you'd be done. Badda bing, badda boom. If you were counting that took me nine mouse clicks which navigated me through three different windows. You might be thinking that wasn't much but just showing you how to do this took up two entire pages from this eBook!

PowerShell

Now, let's see how that can be done with PowerShell.

```
PS C:\> Add-ADGroupMember -Identity 'Domain Admins' -Members 'joe_user'
PS C:\>
```

Done and done. No mousing around, accidentally hitting buttons and navigating through nine different windows. I didn't even use an entire paragraph of this eBook! All it took was 63 keystrokes. How fast can you type? That's as fast as it can be done. Granted, you might not know how to do this off the top of your head but if you keep reading you *will*!

To Automate

You've seen how much faster you can be at getting things done in AD with PowerShell by simply using PowerShell over the traditional GUI but we're not through yet. Automation is what really sets managing AD with PowerShell apart from the GUI.

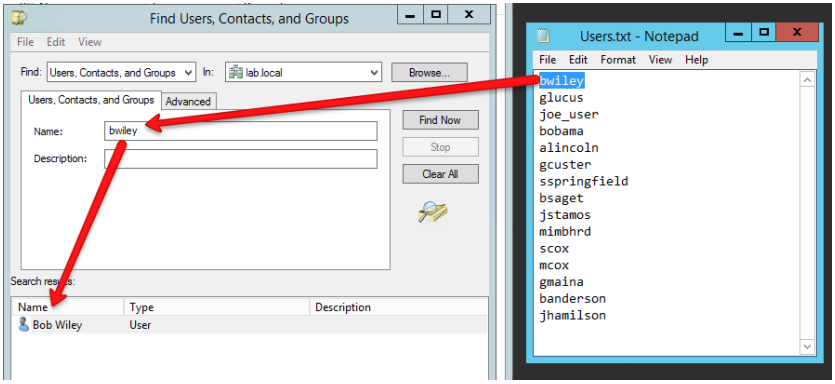
There's been hundreds of times in my career where I've been tasked with making lots of changes in AD at once. I've had to change a bunch of employee IDs on user accounts, move a bunch of computers that were scattered all over the AD tree and consolidate them into a single OU and take a CSV file full of employee information that Human Resources gave me, match each of those rows in the CSV file to an employee's user account and then update a bunch of AD attributes. Is this sounding familiar? Don't tell me you just copied and pasted 'til your fingers were numb on that CSV example. Let me show you the light.

I always think it's best to explain concepts in examples so let's use another example to demonstrate where the real power of using PowerShell with AD lies. Let's say I've got a text file full of employee AD user accounts that are spread out across lots of different OUs. All of these user accounts are for various employees in the same department but the *Department* attribute is not filled in for each. You need to fill in that Department attribute with *Accounting*.

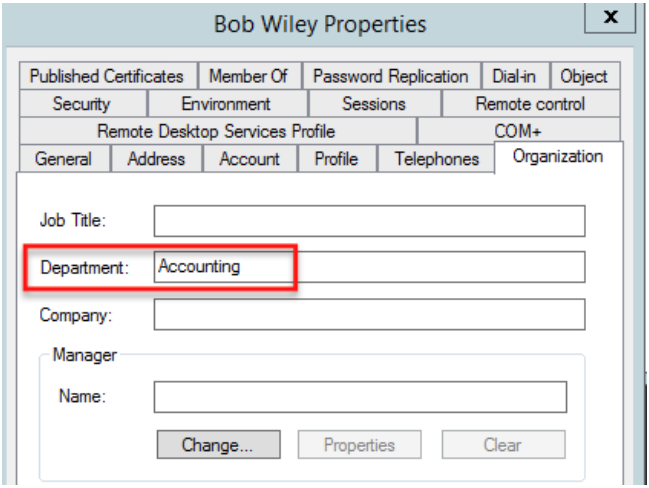
The GUI

I've got fifteen user accounts in my text file that I need to update. Let's see how to do this via the GUI.

1. Copy the user account from the text file to ensure you don't fat-finger it, paste it into the AD find dialog box and double click on the user account to bring up the user properties.



2. Try and remember which tab the department was on the user account properties dialog box, typing in Accounting and hit OK.



3. Repeat, repeat again, repeat again, repeat again and repeat again another ten times. Not only might you fall asleep but what if you get interrupted halfway through and forget your place. Maybe you accidently copy too much from the text file and have to do one another time or maybe...you get the point here. This is tedious, mundane work.

PowerShell

```
PS C:\> Get-Content C:\Users.txt | Set-AdUser -Department 'Accounting'
PS C:\>
```

Done. Using PowerShell I was able to read each of the user accounts in the `C:\Users.txt` file, send all of those users to the PowerShell `Set-AdUser` command and simply tell it to update the Department attribute to be Accounting on each user account. Can it get any simpler than that?

Getting Started

Do I have your attention yet? Are you kicking yourself for not looking into this PowerShell thing earlier? Now that you've got a taste for just a tiny fraction of what's possible managing Active Directory with PowerShell it's now time to take it up a notch. I haven't even shown you "scripting" yet. We've just been doing simple one-liner commands. I have so much to show you.

Some Assumptions

Before we get too far I'm going to make some assumptions here.

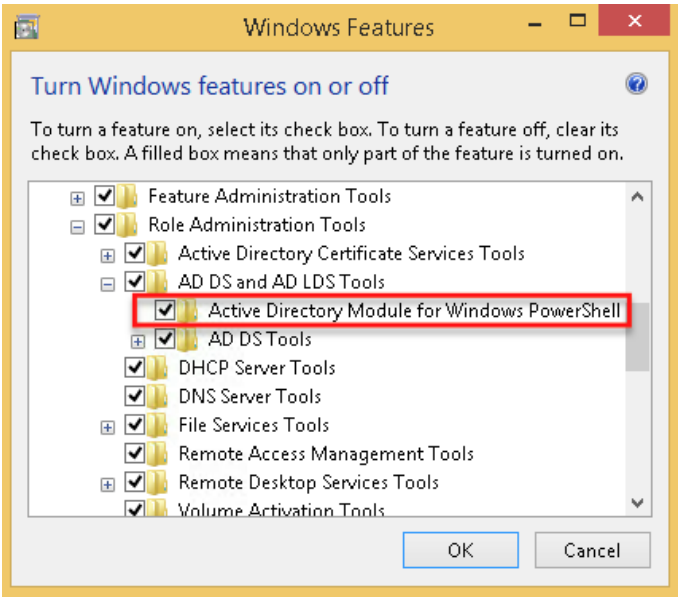
1. You're on Windows 8.1.
2. Your computer is a member of the domain you'd like to script with.
3. You have the rights on the domain to query objects and to write attributes to.

With that out of the way, let's get started!

Prerequisites

Unfortunately, it's not possible to sit down at a freshly built Windows computer domain member and begin scripting AD. You'll first need to get a few things in order. The first is installing the Remote Server Administration Tools (RSAT)

package. Each RSAT package is specific to the operating system version. From here on out I'm going to assume you've got Windows 8.1. The install is a piece of cake. Once you've got the install done you should now have the option to enable *Remote Server Administration Tools → Role Administration Tools → AD DS and AD LDS Tools → Active Directory Module for Windows PowerShell*.



By enabling this feature you now have the Active Directory PowerShell module installed. We now don't need to use this stinkin' GUI any more!

Next, fire up your PowerShell console and we'll make sure the AD PowerShell module is available to you. You should get an output similar to what you see here.

```
PS C:\> Get-Module -Name ActiveDirectory -ListAvailable

    Directory: C:\Windows\system32\WindowsPowerShell\v1.0\Modules

ModuleType Version    Name                                ExportedCommands
---------- -------    ----                                ----------------
Manifest   1.0.0.0    ActiveDirectory                     {Add-ADCentralAcc
```

Once you've got the AD module installed it's always a good idea to run the Update-Help command. PowerShell has a comprehensive help system and provides you with the Update-Help command to ensure the help on your computer never goes stale. By running Update-Help, your computer will download all of the latest documentation for each of the PowerShell commands from Microsoft. This may take a little while at first but it's worth the time especially when you're exploring a new module.

You now have the world in front of you and never need to use the GUI again! ...well, you might still need to use it every now and then but let's pretend that you're a command-line junkie now.

The Active Directory PowerShell Module

Now that you've got the module installed let's go over some basic commands. I showed you an example of those commands; Add-AdGroupMember and Set-AdUser a little earlier in the examples but now we'll dive a little bit deeper into the commands.

When learning any new module in PowerShell your best friend is the `Get-Command` command. This is an extremely useful command that allows you, among many other things, to find all of the commands in a particular module. To do this you simply run `Get-Command` and use the `Module` parameter.

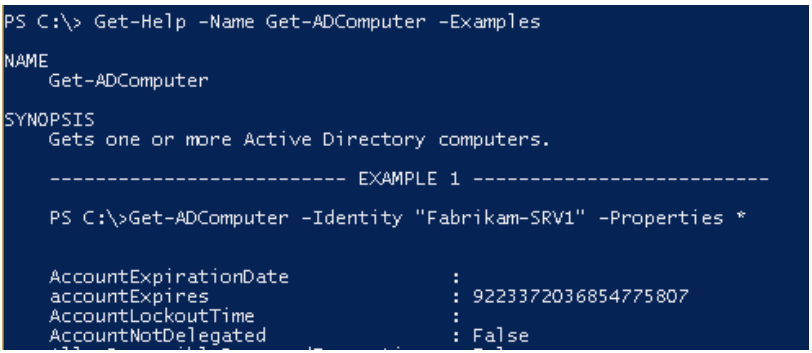
```
PS C:\> Get-Command -Module ActiveDirectory

CommandType     Name                                               ModuleName
-----------     ----                                               ----------
Cmdlet          Add-ADCentralAccessPolicyMember                    ActiveDirectory
Cmdlet          Add-ADComputerServiceAccount                       ActiveDirectory
Cmdlet          Add-ADDomainControllerPasswordReplicationPolicy    ActiveDirectory
Cmdlet          Add-ADFineGrainedPasswordPolicySubject             ActiveDirectory
Cmdlet          Add-ADGroupMember                                  ActiveDirectory
Cmdlet          Add-ADPrincipalGroupMembership                     ActiveDirectory
Cmdlet          Add-ADResourcePropertyListMember                   ActiveDirectory
Cmdlet          Clear-ADAccountExpiration                          ActiveDirectory
Cmdlet          Clear-ADClaimTransformLink                         ActiveDirectory
Cmdlet          Disable-ADAccount                                  ActiveDirectory
Cmdlet          Disable-ADOptionalFeature                          ActiveDirectory
Cmdlet          Enable-ADAccount                                   ActiveDirectory
Cmdlet          Enable-ADOptionalFeature                           ActiveDirectory
Cmdlet          Get-ADAccountAuthorizationGroup                    ActiveDirectory
Cmdlet          Get-ADAccountResultantPasswordReplicationPolicy    ActiveDirectory
Cmdlet          Get-ADAuthenticationPolicy                         ActiveDirectory
Cmdlet          Get-ADAuthenticationPolicySilo                     ActiveDirectory
Cmdlet          Get-ADCentralAccessPolicy                          ActiveDirectory
Cmdlet          Get-ADCentralAccessRule                            ActiveDirectory
Cmdlet          Get-ADClaimTransformPolicy                         ActiveDirectory
Cmdlet          Get-ADClaimType                                    ActiveDirectory
Cmdlet          Get-ADComputer                                     ActiveDirectory
Cmdlet          Get-ADComputerServiceAccount                       ActiveDirectory
```

This will show you all of the various commands available to you. At the time of this writing, there are 147 different commands that do just about anything you can imagine in AD. We can't talk about them all but I encourage you to browse through them.

If you looked through all the commands that are possible with PowerShell you might have gotten a little overwhelmed. That's OK. If you're brand new to PowerShell in general I'm sure you might be thinking what you got yourself into but don't fret! Rome wasn't built in a day. You're not going to up and leave the GUI behind as of right now. Use PowerShell as a companion to the GUI and slowly start trying to replicate what you're doing in the GUI with PowerShell.

Once you find a command that you'd like more information on use the `Get-Help` command.

For each command, PowerShell provides you with a ton of great information on what parameters you can use, what the command does and how to use it. One of my favorite tips is to use the -Examples parameter also. This gives you a few helpful examples of how to use each command and what the outcome will be.



Common Active Directory Tasks

You've downloaded RSAT, got the AD module installed and now should be ready to get started with some common, real-world AD tasks. Because AD is such a big topic, let's start out

with a fairly simple, yet common one; creating new users and getting users into groups.

```
PS C:\> Get-Command -Module ActiveDirectory -Noun '*user*'

CommandType     Name                                               ModuleName
-----------     ----                                               ----------
Cmdlet          Get-ADUser                                         ActiveDirectory
Cmdlet          Get-ADUserResultantPasswordPolicy                  ActiveDirectory
Cmdlet          New-ADUser                                         ActiveDirectory
Cmdlet          Remove-ADUser                                      ActiveDirectory
Cmdlet          Set-ADUser                                         ActiveDirectory


PS C:\> Get-Command -Module ActiveDirectory -Noun '*group*'

CommandType     Name                                               ModuleName
-----------     ----                                               ----------
Cmdlet          Add-ADGroupMember                                  ActiveDirectory
Cmdlet          Add-ADPrincipalGroupMembership                     ActiveDirectory
Cmdlet          Get-ADAccountAuthorizationGroup                    ActiveDirectory
Cmdlet          Get-ADGroup                                        ActiveDirectory
Cmdlet          Get-ADGroupMember                                  ActiveDirectory
Cmdlet          Get-ADPrincipalGroupMembership                     ActiveDirectory
Cmdlet          New-ADGroup                                        ActiveDirectory
Cmdlet          Remove-ADGroup                                     ActiveDirectory
Cmdlet          Remove-ADGroupMember                               ActiveDirectory
Cmdlet          Remove-ADPrincipalGroupMembership                  ActiveDirectory
Cmdlet          Set-ADGroup                                        ActiveDirectory
```

Creating New Users

One of the most common tasks performed in AD is creating new users. If you've dealt with AD for longer than 10 minutes I'm sure you've had to create at least one user account.

Before getting too far into this first real example of scripting Active Directory keep in mind things are always harder the first time. For first timers, especially to PowerShell, this section might seem a little daunting. This is primarily because you might be learning a little PowerShell as a language along with how PowerShell interacts with Active Directory. But don't get discouraged! Just know that this first section will, by far, be the longest because I'll be building the foundation for you. Once you understand this first example, I assure you the piece will easily fall into place later on.

You're probably so used to create users via the GUI you may have not realized that it's rare the little wizard that you walk through is the only process that's required to provision a new user account. You're always going back into the account and setting properties that aren't available in the GUI like the department, manager, phone number, login script, yadda yadda. The New User wizard was meant to get a user object created. It wasn't meant to give you everything necessary to provision a new employee's user account. This is a big difference in the GUI and with PowerShell. You'll find that you aren't limited to the design considerations of the GUI but rather you can create your own method of creating new user accounts.

To create a new user account with PowerShell you'd use the New-AdUser command. As soon as you run Get-Help New-AdUser and take a look at the SYNTAX area where all the parameters are shown you'll soon get an idea of the power you have at your fingertips.

Let's go over a good example. Let's say you've got a new employee that was just hired. This employee's name is David Jones. David's a college student only here for the summer in the Marketing department. David needs a typical home folder like everyone else, should have the usual default password that everyone does and should change that password the moment he logs on.

Just by reading the requirements it looks like we're going to need to fill in a few attributes for this new user account.

- First Name
- Last Name
- Account Expiration Date (David is only here for the summer)
- Department
- Home Folder

- Password
- Username
- Change Password at Next Logon

I'll bring up the help for New-AdUser and this time use the -Parameter parameter. and see if I can figure out which (if any) parameters match up to the attributes I need to assign to his new account. The Parameter parameter (no, that's not a typo) shows you all the parameters you can use and an explanation about what each one does. Since we're not sure which parameter to use I'll use an asterisk to represent all of the parameters to New-AdUser.

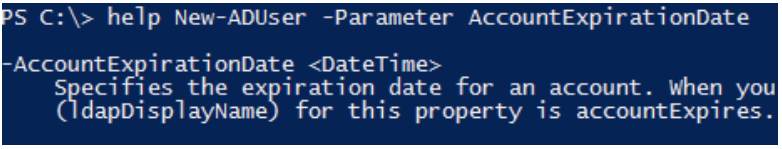
```
help New-AdUser -Parameter *
```

```
PS C:\> help New-ADUser -Parameter GivenName

-GivenName <String>
    Specifies the user's given name. This par
```

By browsing through the parameters available, I was able to find one called GivenName and Surname. These look like they're exactly what I'm looking for to specify David's first and last name.

```
PS C:\> help New-ADUser -Parameter Surname

-Surname <String>
    Specifies the user's last name or surna
```

Next, we'll need to figure out where to specify that we need the account to expire. How about AccountExpirationDate? That seems logical.



Next, we'll find the Department, `HomeDirectory`, `ChangePasswordAtLogon`, Name and finally `AccountPassword` parameters. See a pattern here?

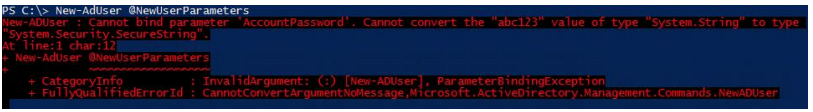
I've now discovered all of the parameters to New-AdUser I need to create this user account. Since we need so many parameters to create this user account I've chosen to use an alternative method called splatting in PowerShell. This is just a funny name for putting all of your parameters and values into a big variable before you run the command. When you need to run the command you'd pass all of those parameters to the command with an @ sign.

```
$NewUserParameters = @{
     'GivenName' = 'David'
     'Surname' = 'Jones'
     'AccountExpirationDate' = '8-7-2015'
     'Department' = 'Marketing'
     'HomeDirectory' =
'\\server\users\djones'
     'AccountPassword' = 'p@$$w0rd10'
     'Name' = 'djones'
     'ChangePasswordAtLogon' = $true
}


New-AdUser @NewUserParameters
```

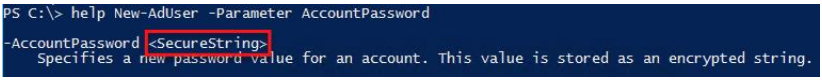
This is exactly the same as the one line below. Which is easier to read? I thought so.

```
New-ADUser -GivenName 'David' -Surname
'Jones' -AccountExpirationDate '8-7-2015'
-Department Marketing -HomeDirectory
'\\server\users\djones' -AccountPassword
'p@$$w0rd10' -Name 'djones' -
ChangePasswordAtLogon $true
```

I've now got my command and parameters to that command all set so I'll go ahead and execute it.



Red text. That's never good. It looks like PowerShell has a problem with the way we used the `AccountPassword` parameter. Let's refer to help again to get some information about this parameter. Hmm...beside the `AccountPassword` parameter there's something called `<SecureString>`. If you look at most of the other parameters you'll just see `<String>` in that same place.



It looks like this parameter might be different. We can't just tell `New-AdUser` to use our default password of `abc123`. It looks like we're going to have to do something different with it. After a little bit of searching online I found that AccountPassword cannot just accept a plain text string like abc123. It actually only takes a secure string. By using the `ConvertTo-SecureString` command and wrapping it in parentheses and then using the output of *that* command `New-AdUser` now likes all of the parameters we used.

```
$NewUserParameters = @{
    'GivenName' = 'David'
    'Surname' = 'Jones'
    'Name' = 'djones'
    'AccountExpirationDate' = '8-7-2015'
    'Department' = 'Marketing'
    'HomeDirectory' =
'\\server\users\djones'
    'AccountPassword' = (ConvertTo-
SecureString 'p@$$w0rd10' -AsPlainText -
Force)
    'ChangePasswordAtLogon' = $true
}

New-AdUser @NewUserParameters
```

If this is now run you'll soon see a `djones` user account pop up in the Users container.

If you're thinking to yourself right now, "Adam, this is definitely *not* easier than the GUI. I can't remember all this stuff!", I understand. This is the hardest part in learning how to use PowerShell for the first time with a new product. Once you've gotten past this it's now just a matter of switching out values. You've already built the framework. Need to create another college student user account only this time it's for Susie Shade? Just replace the values in for `GivenName`, `Surname`, `Name` and `HomeDirectory`. You will create the exact same user account every time.

Adding New Users to Groups

You've now got a new user account created. Great! Now what? Maybe you're done but probably not. Did you forget to put David in the Marketing group? Oh no! Never fear, PowerShell is here so you don't have to remember anymore!

Even though there's no `-Group` parameter on the `New-AdUser` command doesn't mean we can't make it happen. We'll just have to put another block on this little user provisioning script we're creating. One of the beauties of PowerShell is how easy it is to bolt on functionality. This is a great example.

There are some functions (`New-AdUser` being one of them) that, when run successfully, don't output anything at all. You're just taken back to the familiar PS:> prompt. This typically isn't a problem unless you're not done yet. If `New-AdUser` returns nothing at all there's no way to do anything with that user account. Enter the `-PassThru` parameter. The `-PassThru` parameter is a common parameter on numerous commands in PowerShell that tells the command to override it's default behavior and return the object that it normally silences. In our example, it's a user account object.

I'm going to add the `-PassThru` parameter to the `$NewUserParameters` variable from above to `New-AdUser` and run the exact same command I ran earlier and see what happens.

```
PS C:\> $UserAccount = New-AdUser @NewUserParameters
PS C:\> $UserAccount


DistinguishedName : CN=djones,CN=Users,DC=lab,DC=local
Enabled           : False
GivenName         : David
Name              : djones
ObjectClass       : user
ObjectGUID        : 9fe853d2-0486-4b4c-b87c-77c202728694
SamAccountName    : djones
SID               : S-1-5-21-3353910224-3636413190-1920695484-1742
Surname           : Jones
UserPrincipalName :
```

You'll see that instead of just running `New-AdUser @NewUserParameters` I decided to assign the output (since we've got some now) to the variable `$UserAccount` and show that, in fact, we have captured the user account object showing lots of great information about the user account that I just created.

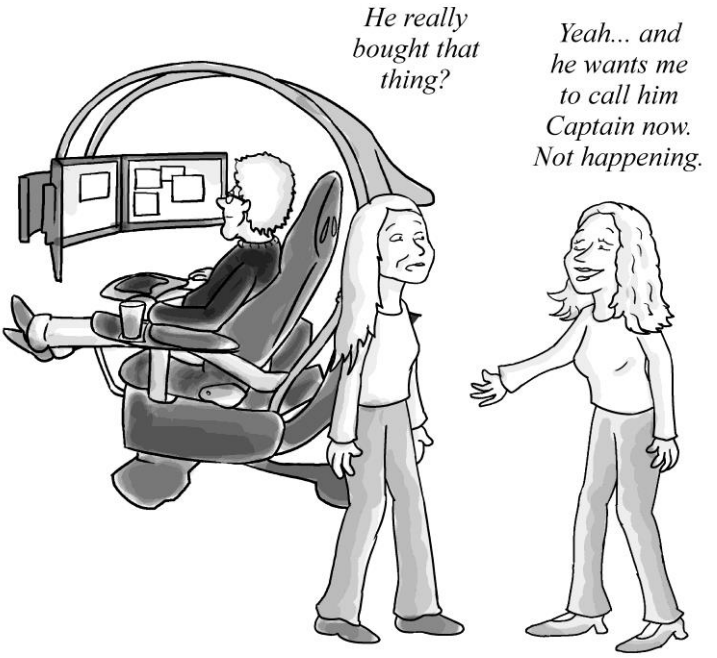
That's all well and good but I'm still not doing anything different other than creating a user account. I want to add this user account to a group. Luckily, we have `Add-AdGroupMember` just for this purpose.

```
PS C:\> $UserAccount = New-AdUser @NewUserParameters
PS C:\> Add-AdGroupMember -Identity 'Marketing' -Members $UserAccount
PS C:\> Get-ADPrincipalGroupMembership -Identity $UserAccount


distinguishedName : CN=Domain Users,CN=Users,DC=lab,DC=local
GroupCategory     : Security
GroupScope        : Global
name              : Domain Users
objectClass       : group
objectGUID        : 7e96acf9-ef6a-4e4a-b122-857c74f4ce4b
SamAccountName    : Domain Users
SID               : S-1-5-21-3353910224-3636413190-1920695484-513

distinguishedName : CN=Marketing,DC=lab,DC=local
GroupCategory     : Security
GroupScope        : Global
name              : Marketing
objectClass       : group
objectGUID        : 726d7b88-c597-4a2b-9492-d61b568e70b2
SamAccountName    : Marketing
SID               : S-1-5-21-3353910224-3636413190-1920695484-1743
```

Now that something's actually in `$UserAccount` I can use `Add-AdGroupMember` to follow right up after it and add the newly formed account to the Marketing group. I even added in a free demonstration of `Get-AdPrincipalGroupMembership` to show you I wasn't lyin...or….how to use a new command!

Once you get past the "ewww...command line" stuff, do you see the power here? Take what I'm showing you and multiply the example complexity by 100. By using PowerShell to accomplish tasks like this allows to automate what was once a long, tedious and drawn-out process into a simple, straightforward task that now takes minutes or less!

Building A Script Not Just Running a Command



You've officially been indoctrinated into the AD/PowerShell club. As part of your membership dues you now must undertake a project your GUI-toting peers shudder at the thought of with nothing more than a PowerShell console! Are you ready for your next challenge?

In this chapter I'm going to let go of the bike seat a little and let you ride on your own. Don't worry. I'll be right behind you the whole way.

Keeping Active Directory Clean

AD has a tendency to get "dirty". The amount of filth that accumulates and how fast the grime turns out depends on the size and change frequency of object in AD. I'm positive you've spotted computer accounts that existed for employees that have been fired years ago. You've probably manually removed your fair share of GPOs that have been stagnate for years not being linked to any OU at all. Or, my favorite, are all the bsmith, bsmith1, bsmith2, bsmith3 user accounts that exist because of Bob Smith that was fired 5 years ago still has an account but no one deletes so when Barb Schumaker, Brent Stallings and Brody Stens are hired they simply get a number.

In this final chapter, I'm going to leave you with a script that you can take with you and immediately begin using it in your AD environment I'm calling your AD Cleaner script. Without further ado, let's jump right in.

Before undertaking any sizable script it's always a good idea to outline what you intend to do and how you're going to do it. I'm going to "clean" AD isn't too specific and the last time I checked I didn't see a "Clean-Ad" PowerShell command that magically did everything you wanted. We need to break down the functions we intend to do.

1. Find all disabled computer accounts.
2. Find all user accounts that haven't logged in in 90 days.
3. Find all disabled GPOs.
4. Find all groups that don't have any members in them.
5. Find all GPOs that aren't linked to an OU.

6. Find all user accounts that have expired their password 30 days ago or more.

Do you think you're up to the task? To start, I'm going to create a PS1 file with a descriptive name of Get-AdJunk.ps1. I'll be adding all of my code to this file.

Disabled Computer Accounts

First up is finding all of the disabled computer accounts. Pftt..child's play. Using the `Search-AdAccount` I can easily find all of the computers in my domain that are disabled. Not even a challenge.

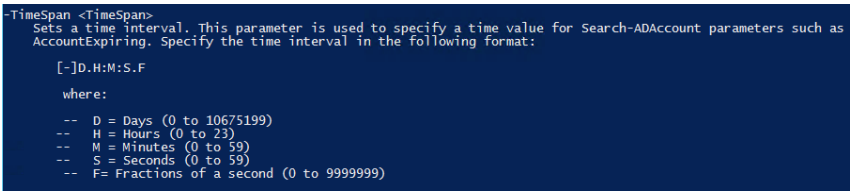
```
Search-AdAccount -AccountDisabled -
ComputersOnly
```

Inactive User Accounts

Again, using the handy `Search-AdAccount` command this is not a problem at all to do. Speaking of, I highly recommend checking out all the parameters that this cmdlet has. It is extremely useful in cleanup scripts like we're creating in a number of different scenarios.

```
Search-AdAccount -AccountInactive -
Timespan 90.00:00:00
```

The `-Timespan` parameter might look at little off. The reason is because this parameter only accepts TimeSpan input. Remember to use `Get-Help` with the `-Parameter` parameter to read all about it!



Disabled GPOs

Unfortunately, we can't use the handy Search-AdAccount command here. However, continuing with the intuitive naming convention that PowerShell uses how do you think you'd find a GPO? `Get-GPO`, maybe? Ding! Ding!

`Get-GPO` has a parameter called `-All` which is used to find all of the GPOs in your domain. But we're not interested in all of them. I just want to see the ones that are disabled.

Each GPO that `Get-Gpo` finds has a property called `GpoStatus`. This property indicates if the user settings, computer settings or both are enabled or disabled.

```
PS C:\> Get-GPO -All


DisplayName      : Default Domain Policy
DomainName       : lab.local
Owner            : LAB\Domain Admins
Id               : 31b2f340-016d-11d2-945f-00c04fb984f9
GpoStatus        : AllSettingsEnabled
Description      :
CreationTime     : 9/27/2014 1:18:16 PM
ModificationTime : 7/17/2015 2:58:04 PM
UserVersion      : AD Version: 2, SysVol Version: 2
ComputerVersion  : AD Version: 33, SysVol Version: 33
WmiFilter        :

DisplayName      : Default Domain Controllers Policy
DomainName       : lab.local
Owner            : LAB\Domain Admins
Id               : 6ac1786c-016f-11d2-945f-00c04fb984f9
GpoStatus        : AllSettingsEnabled
Description      :
CreationTime     : 9/27/2014 1:18:16 PM
ModificationTime : 7/15/2015 5:07:20 PM
UserVersion      : AD Version: 2, SysVol Version: 2
ComputerVersion  : AD Version: 13, SysVol Version: 13
WmiFilter        :
```

We're interested in only the ones that show `AllSettingsDisabled`. Using the ubiquitous `Where-Object` command which filters the output of `Get-GPO` we can narrow down only the GPOs that have both user and computer settings disabled.

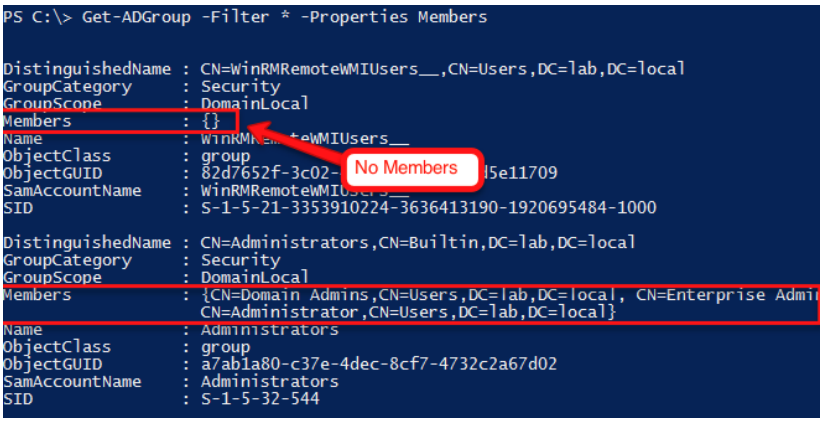
```
Get-GPO -All | Where-Object { $_.GpoStatus
-eq 'AllSettingsDisabled' }
```

No Member Groups

Groups with no members in them aren't doing much good so we need to see which groups are just wasting space in AD. Going with the same `Get-Ad*` naming convention, we will use `Get-AdGroup` for this. `Get-AdGroup` has a `-Filter` parameter. You'll see this one a lot of the AD commands. This is used to filter out any objects from being returned that meet a specified criteria. Although, the preferred method to filter objects, it's not possible in this case so we'll need to use `Where-Object`.

You can see that each of the group objects that `Get-AdGroup` returns has a `Members` property that includes all of members inside of that group. We want to only see group that have 0 members so we can use the `Count` property that exists on collections like this. In this instance, I'm telling PowerShell that I want to see all groups where the `Members` property has no elements inside of it.

```
Get-ADGroup -Filter * | Where-Object {
$_.Members.Count -eq 0 }
```



GPOs Not Linked to Anything

We're now ready to take the next step and come away from the one-liner. To find all GPOs that aren't linked to anything requires a little more advanced approach. This time we're going to have to mess with a little XML. XML is traditionally known to be tough to wrangle incode but PowerShell provides this great feature called an XML type which a bunch of text that represents an XML file can be cast to. Casting is just a term used to signify that you want to turn some variable into another kind of variable.

This part of the script has three distinct parts:

1. Running `Get-GPOReport` to find all of the GPOs represented in XML format.
2. Casting the output of all `Get-GPOReport` to a XML type.
3. Enumerating all of the GPOs represented inside `$Gpos` to find only the ones that have a `LinksTo` property.

First up, we'll run `Get-GPOReport` with a couple parameters; `-All` and `-ReportType`. Similar to the `Get-GPO` command, the `-All` parameter is used to find all of the GPOs; not just one. Some PowerShell commands require certain parameters; ReportType is one of those on the `Get-GPOReport` command. `Get-GPOReport` allows you to output GPOs in a HTML or XML format. Since we're not concerned about actually looking through the GPOs we need XML to easily filter on that `LinksTo` property later.

```
$Gpos = Get-GPOReport -All -ReportType XML
```

This is great! We've captured all of the GPOs in an XML format but there's a problem. As is, there's no way to easily figure out if each of those GPOs has the `LinksTo` property we're looking for.

```
PS C:\> $Gpos = Get-GPOReport -All -ReportType XML
PS C:\> $Gpos
<?xml version="1.0" encoding="utf-16"?>

<GPOS>

<GPO xmlns:xsd="http://www.w3.org/2001/XMLSchema" xmlns:xsi="http://www.w3.org/2001/XMLSchema-instance" xmlns="http://www.microso
ft.com/GroupPolicy/Settings">
  <Identifier>
    <Identifier xmlns="http://www.microsoft.com/GroupPolicy/Types">{0F728F22-07A0-4F1C-A706-0D6DD06FECFD}</Identifier>
    <Domain xmlns="http://www.microsoft.com/GroupPolicy/Types">lab.local</Domain>
  </Identifier>
  <Name>New Group Policy Object</Name>
  <IncludeComments>true</IncludeComments>
  <CreatedTime>2015-08-01T20:56:54</CreatedTime>
  <ModifiedTime>2015-08-01T20:56:54</ModifiedTime>
  <ReadTime>2015-08-01T23:25:48.7111992Z</ReadTime>
  <SecurityDescriptor>
    <SDDL xmlns="http://www.microsoft.com/GroupPolicy/Types/Security">O:DAG:DAD:PAI(OA;CI;CR;edacfd8f-ffb3-11d1-b41d-00a0c968f939
```

Good luck trying to figure anything out in that enormous string! We need to cast that `$Gpos` variable to an XML type so we can look for the `LinksTo` property just like we have previously with other commands.

```
[xml]$Gpos = Get-GPOReport -All -
ReportType XML
```

Do you see the slight difference? It's the `[xml]` added to the front of `$Gpos`. This tells PowerShell that `$Gpos` contains XML content and to transform that XML content into something humanly readable. We can now reference `$Gpos` like we have all of the other objects using dot notation.

```
PS C:\> $Gpos.GPOS

GPO
---
{New Group Policy Object, Default Domain Policy, Default Domain Controllers Policy, User Logon/Logoff Auditing}
```

Are you still with me? Mentally, I mean. Your brain's not fried yet? Well, let's keep truckin'!

Finally, we're now able to search for that `LinksTo` property I've been talking about all along. Again, using the `Where-Object` command, I can simply filter out all of the GPOs that don't have that property and which only outputs the GPOs that are not linked to anything!

```
$Gpos.GPOS.GPO | Where-Object {-not
$_.LinksTo}
```

User Accounts with Old, Expired Passwords

We've come to the final example of the AD Cleaner script. If you've made it this far you deserve a medal. I know since you're new to AD and PowerShell this command-line stuff might have been a culture shock but I hope you're able to see past all the keystrokes and see the big picture. Who knows. You might get good enough one of these days and ditch the GUI for good!

With that being said, let's get this eBook wrapped up and get you on your way.

As with the other parts of the script that are more complicated than a simple one-liner it's important to first briefly plan out what your code is going to do and the blanks you're going to have to fill in when it comes time to write code.

1. How "old" is "old"? 30 minutes, 30 hours, 30 days?
2. How do I find all users in that have an expired password?
3. To find how long a user account has been in the `PasswordExpired` state, we're going to need to find the date that the password was expired. How?
4. How are we going to compare those two dates to get the age?

First, I want to know all user accounts that have had their password expired for more than 30 days. Question #1 is done. That was the easy one.

Next, finding user accounts that have expired passwords is easy with the `Search-AdAccount` command.

```
Search-AdAccount -PasswordExpired -
UsersOnly
```

But this is the last example. You don't think I'd let you off *that* easy do you? I want to know how long each of those accounts have been in that state. This is going to require some thinking. On to question #3.

Question #3 is going to be a little harder to answer. After sleuthing around a little bit I found that there's a `PasswordLastSet` attribute on all user accounts. This is an attribute that shows the date and time when the password was either created for the first time or changed. We now know when a password was updated but I still don't know when it actually expired. Or do I? We've got a password policy that forces all users to change their password after a certain number of days. If I know that, I could just take the `PasswordLastSet` date and add the number of days that's in the password policy which will give me the exact date and them when the password expired!

One of the aspects of scripting is how it forces you to think. Scripting forces you to break down a problem into little pieces. This, inherently, forces you to understand the problem at a much deeper level than if you were simply clicking through some windows and letting the GUI handle pieces for you. But can I get that in PowerShell? Of course!

```
$MaxPasswordAge = (Get-
ADDefaultDomainPasswordPolicy).MaxPassword
Age.Days
```

Once you've got the number of days after a password was changed, you can now easily calculate the time the password has expired.

PasswordLastSet + *MaxPasswordAge* = *PasswordExpiredDate*

Here's a code snippet that will show you when each user account was expired. Even though `Search-AdAccount` returns some properties of users with expired passwords it doesn't return the `PasswordLastSet` property we're looking for. To get this, we're forced to use `Get-AdUser` to do a second lookup (inefficient yet necessary code *this* time). This is nice, but we need to see user accounts that are older than this date.

```
Search-AdAccount -PasswordExpired -
UsersOnly | ForEach-Object { (Get-AdUser -
Filter "samAccountName -eq
$_.SamAccountName").PasswordLastSet }
```

Now comes the fun question; question #4. This is how we put this all together. At this point, I've got all the user accounts that have expired passwords and I know when those accounts have expired. It's now time to figure out how old that particular date is. Remember, we're only interested in ones that are 30 days or older. We must calculate the difference in days from today's date to the "PasswordExpiredDate". Introducing the code we're looking for:

```
$InactiveDays = 30

$MaxPasswordAge = (Get-
ADDefaultDomainPasswordPolicy).MaxPassword
Age.Days

$InactiveThreshold = $MaxPasswordAge +
$InactiveDays

Search-AdAccount -PasswordExpired -
UsersOnly | Where-Object {((Get-Date) -
(Get-AdUser -Filter "samAccountName -eq
$_.SamAccountName").PasswordLastSet) -lt
$InactiveThreshold}
```

This may be a little hard to digest at first so here's a breakdown:

- `$InactiveThreshold` is the latest date that we'd want to see a password expired on.
- All users with expired passwords are returned by: `Search-AdAccount -PasswordExpired -UsersOnly`
- In the `Where-Object`'s curly braces `{ }`, `$_.SamAccountName` is means the current username it's working on.

- The current username's `PasswordLastSet` attribute if found with this command: `(Get-AdUser -Filter "samAccountName -eq $_.SamAccountName").PasswordLastSet`
- The `-lt` is an operator that means "less than". In this instance, being in the `Where-Object` block, it's saying only show me user accounts where their password was set before the threshold we care about.

To give you an example, today is 08/01/15 and the maximum time you allow a password to be in use before it's expired is 60 days. You've got two user accounts; one that changed their password 120 days ago and one that changed their password 70 days ago. They both haven't changed it since thus they're both expired.

UserA

PasswordLastSet = 04/03/15

PasswordExpiredOn = 04/03/15 + 60 (Max password age) = 06/02/15

User A's password has been expired for 60 days

UserB

PasswordLastSet = 05/23/15

PasswordExpiredOn = 05/23/15 + 60 (Max password age) = 07/22/15

User B's password has been expired for 10 days

You've now found all of the "junk" in AD. I now pass the baton to you to figure out how to remove it.

The Big Takeaways

You've just been taken from the bare essentials of scripting Active Directory with PowerShell to a fairly complex task. You've no doubt been a little confused, taken back, had various "ah ha!" moments, gotten excited and frustrated at the same time. All in all, you've seen what it takes to begin scripting Active Directory.

During your journey, I hope you've had countless lightbulbs go off above your head with new ideas and things you could try in your environment. I encourage you to not just stop here. We've just scraped the surface as to what's capable with PowerShell and Active Directory.

Automation naturally leads itself to eliminating human error and saving tons of time down the road. One aspect that's not inherently obvious is that scripting really makes you learn a topic; not just the code syntax but how a topic works. Wherever you go in AD, PowerShell will follow and if you choose to use it, you will get frustrated, throw your hands up in disgust and want to quit. If you persist, however, and continue scripting AD, you'll come away with a whole new understanding AD objects, the relationships between those objects and how AD can be so resilient in providing a robust single sign on solution.

Talk about what's possible with PowerShell and Active Directory.

Active Directory is in just about every organization. It's a product that's used to verify you are who you say you are among many other authentication activities.

The goal of this book is to introduce newcomers to Active Directory that may know just a little about it but want to know more about automating tasks around Active Directory. It will show the reader that they can be much more productive if they take a little time, learn a little PowerShell and really see how to make Active Directory hum.

About Adam Bertram

Adam Bertram is an independent consultant and specializes in consulting and evangelizing all things IT automation. Adam is a Microsoft Windows PowerShell MVP and has numerous Microsoft IT pro certifications. He is a writer, trainer and presenter and authors many online courses. He's a regular contributor to numerous print and online publications and presents at various user groups and conferences. You can find Adam at adamtheautomator.com or on Twitter at @adbertram.