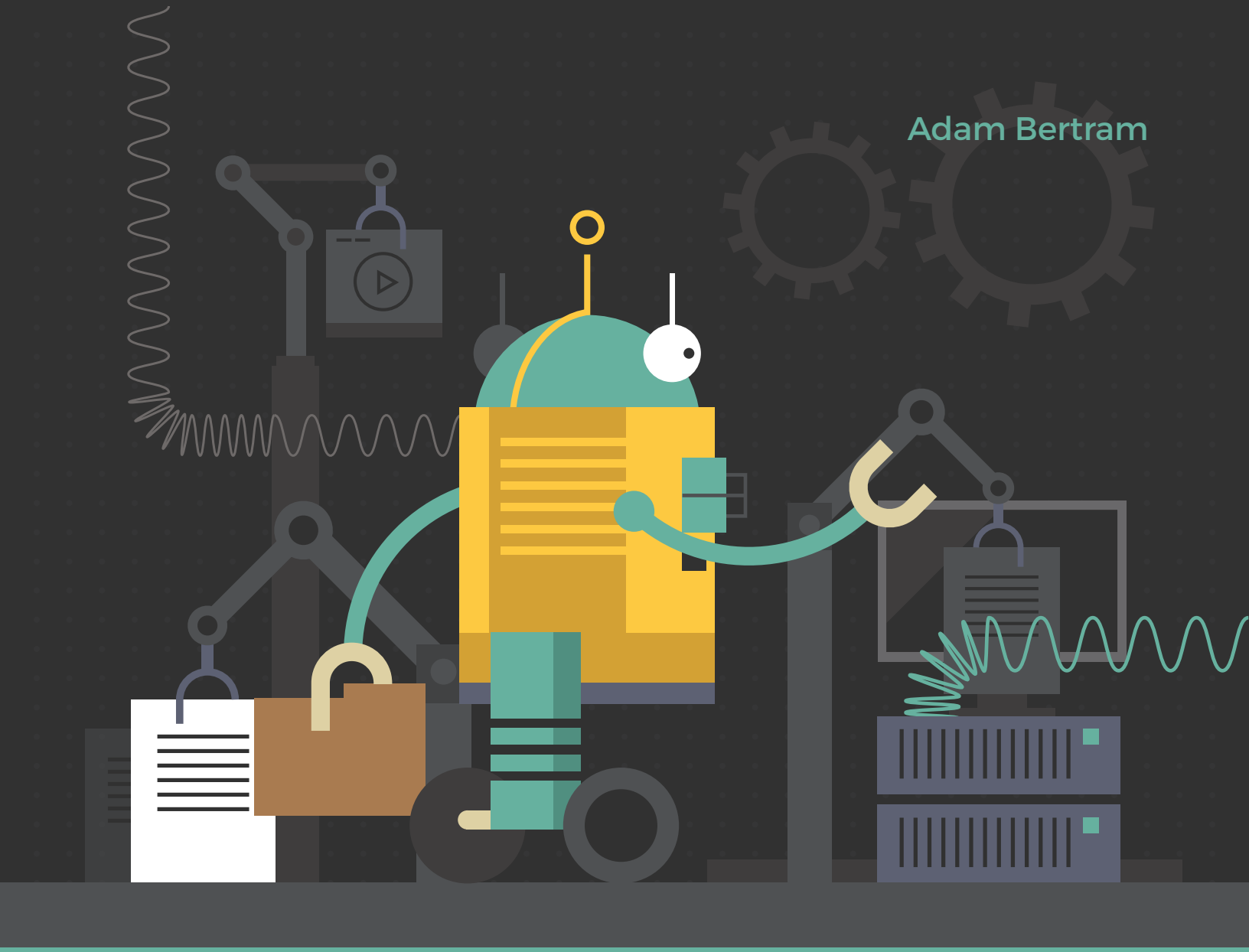
Adam Bertram



How to Automate Using PowerShell

How to Automate Tasks, File Transfers, and Data Security

ipswitch

Introduction

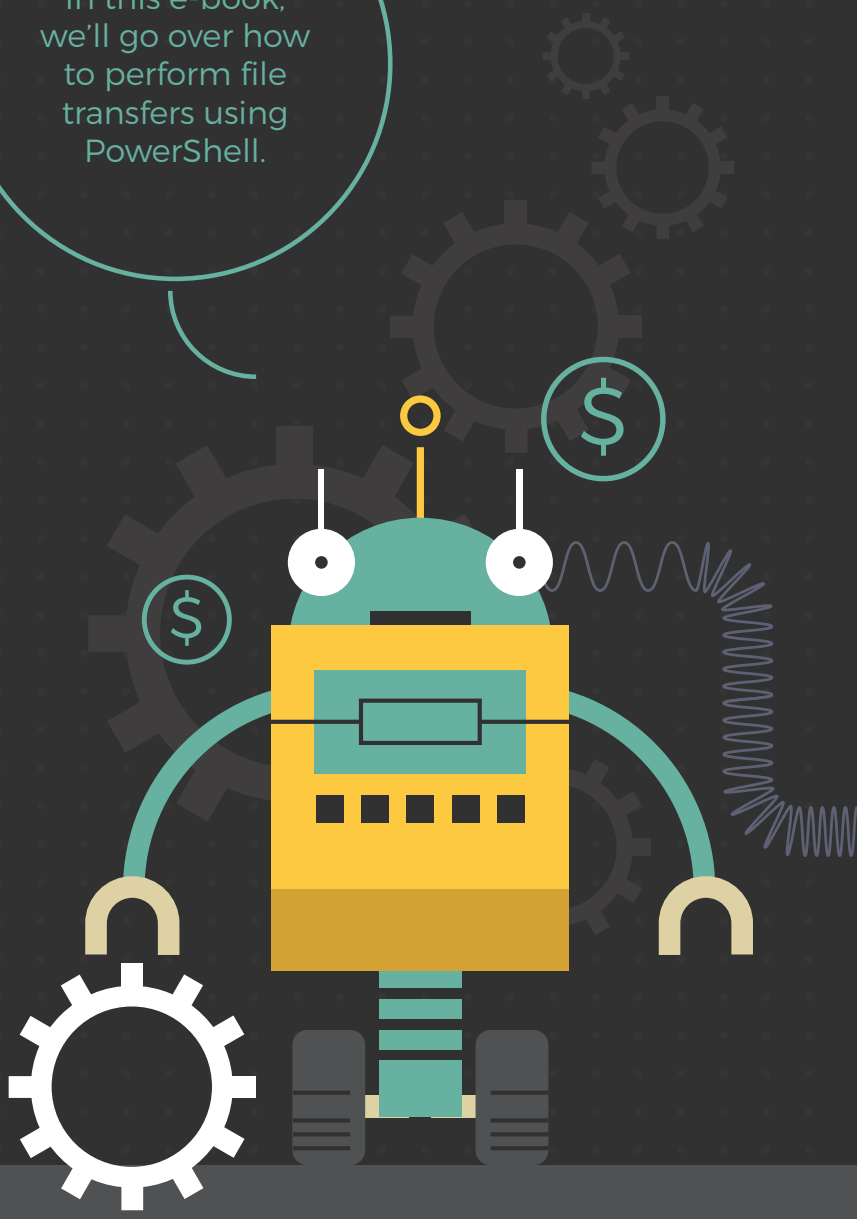
Increased worker output is on the wish list of every organization out there. However, with budgets tightening and work demands increasing, this can prove challenging. IT is already stretched thin as it is. How do organizations continue to keep up while minimizing mistakes, ensuring the results meet the user's needs and staying within budget? The answer is automation and PowerShell.

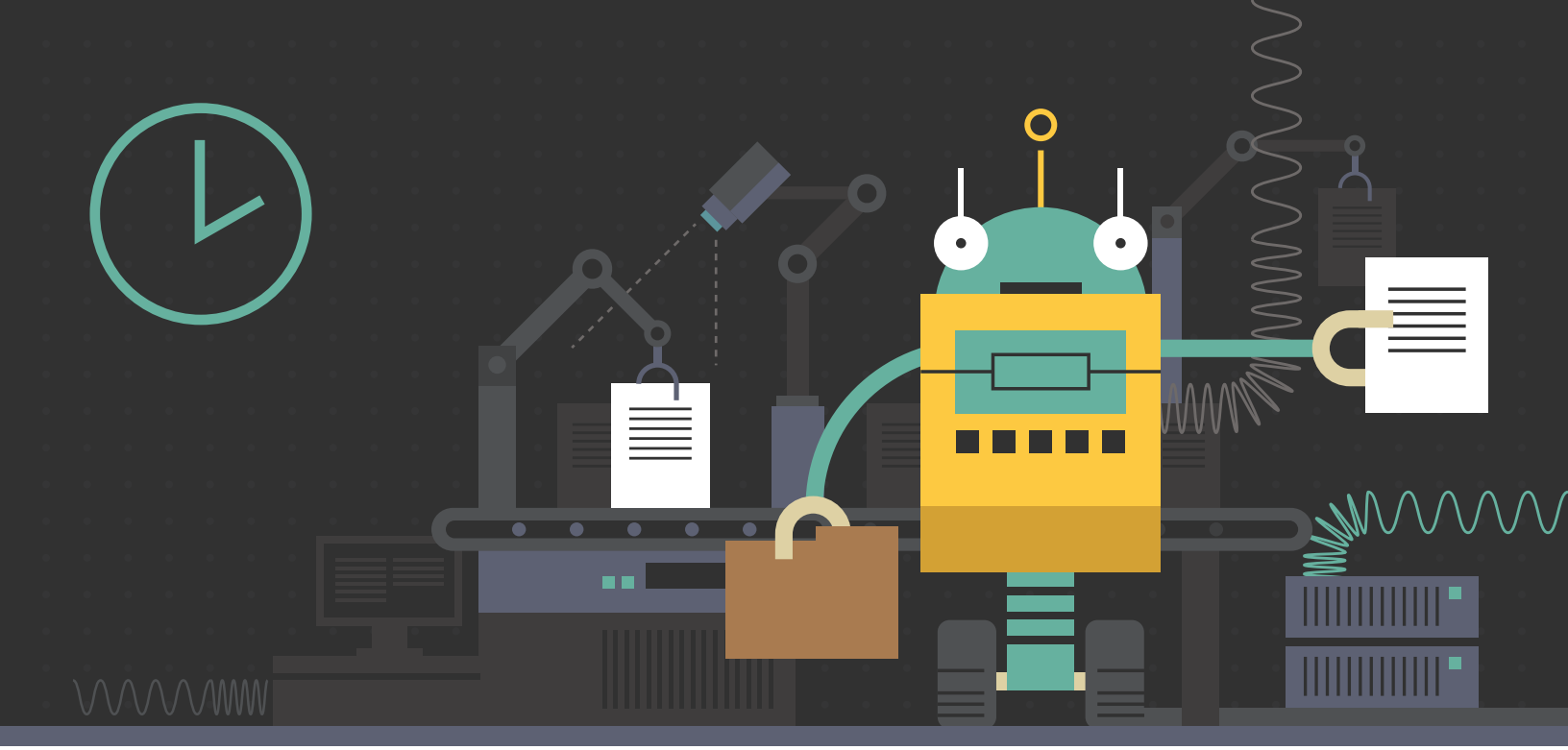
IT is slowly becoming a business asset to organizations rather than a utility. As such, it's essential for IT to automate as many processes as possible to get workers back to doing what humans do best: coming up with creative solutions to problems.

One common task that's ripe for automation is file transfer. Data is stored in a multitude of different files, and IT orgs sometimes deal with millions of them. These files are then stored on premise, in the cloud and are transferred to other organizations and more, all on an ongoing basis. Most of the time, these file transfers are predictable. A department might need a report at a particular time each day; a business partner might need the latest Excel spreadsheet detailing new product specifications, or a database might need to be backed up in the cloud to a file.

What's a good way to automate file transfers? One way is building out this automation on your own with PowerShell, which gives IT the power to automate any file transfer task they wish. In this e-book, we're going to cover just a few applications of scripts to help the IT worker save some time and get back into problem-solving mode.

In this e-book, we'll go over how to perform file transfers using PowerShell.





How to Use Scheduled Tasks to Automate File Transfers

Every file transfer has a trigger. That trigger can be ad hoc which means the file is moved when an IT worker performs some action or it can be automatic. In this article, we'll cover how to use PowerShell to create scheduled tasks that will automate a file transfer script.

Copying files from one place to another is a trivial task no matter how you do it. And there are a number of ways to get the job done: dragging and dropping the file in Windows Explorer, Copy-Item with PowerShell or the simple copy command in DOS. It's just a matter of specifying a source and a destination path and setting a few other optional parameters. It's only when you start copying a lot of files on a frequent basis that you run into problems. You shouldn't have to babysit all of the file copies; scheduled tasks is perfect for automating this job.

When automating file copies, especially in a Windows environment, your go-to scripting language is going to be Windows PowerShell. If you need to quickly copy one or more files from one destination to another, PowerShell is a great way to do that. Also, not only is it easy to manually kick off PowerShell scripts, but you can also trigger transfers via PowerShell scripts by using Windows scheduled tasks.

In this article, we'll go over how to perform file transfers using PowerShell by writing a script and creating a scheduled task to kick off that script on a recurring basis. But before we start, I'm going to assume that you have at least PowerShell v4 installed on your computer. Otherwise, the tricks I'm about to show you may not work properly.

Create Your Script

First you need to create a script to perform file transfers. Let's call the script CopyFiles.ps1. This script will contain the following code:

```
param(

[string]$SourcePath,

[string]$DestinationPath

)

Copy-Item –Path $SourcePath
–Destination $DestinationPath -Recurse
```

As you can see, the script is simple, but it leaves room for lots of customization depending on your environment.

The most complicated part of this script is the param() section. This is a parameter block containing two parameters: SourcePath and DestinationPath. By making both of these values, parameters allows us to pass in different values to our script so we can reuse it. If SourcePath and DestinationPath were actual paths, we'd have to create separate scripts for every different file copy!

The most complicated part of this script is the param() section.

Manually kicking off this script will look something like this:

```
& .\CopyFiles.ps1 –SourcePath C:\Source
–DestinationPath \\SERVER\Destination
```

This example would copy all files and subfolders in the C:\Source folder to the \\SERVER\Destination shared folder.

Create a Scheduled Task

Now that you have your CopyFiles.ps1 PowerShell script, head over to the computer where you'd like to kick it off. In this example, we're going to create a scheduled task to run this script once a day at 3 a.m.

You could create scheduled tasks by running the Task Scheduler GUI and creating one that way, but we're all about automation here. Let's learn how to create the scheduled task in PowerShell as well. To do this, you'll need to complete four rough steps:

1) Create the scheduled task action.
2) Create the trigger.
3) Create the scheduled task in memory.
4) Create the scheduled task on the computer.

Here's what that looks like in practice. First, we'll create the scheduled task action. This defines the EXE to run along with any arguments. Here, I'm assuming that your script is located at C:\CopyFiles.ps1.

```
$Action = New-ScheduledTaskAction
-Execute
'C:\Windows\System32\WindowsPower
Shell\v1.0\powershell.exe' -Argument
"-NonInteractive -NoLogo -NoProfile -File
'C:\CopyFiles.ps1' –SourcePath 'C:\Source
–DestinationPath '\\SERVER\Destination'"
```

Next, we'll create a trigger to kick it off at 3 a.m. every day.

```
$Trigger = New-ScheduledTaskTrigger
-Daily -At '3AM'
```

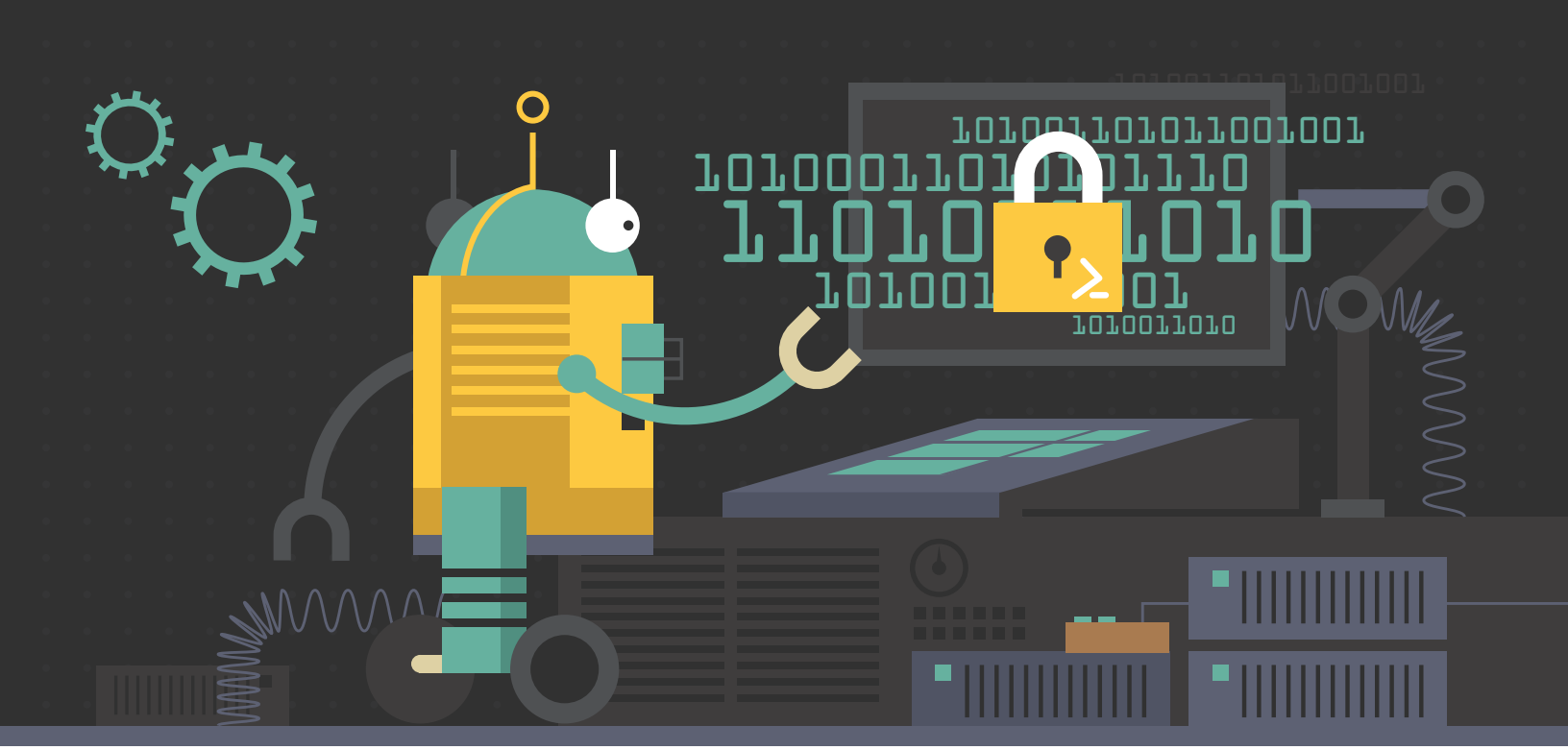
Next, we'll create the scheduled task in memory using the action and trigger that we just created.

```
$Task = New-ScheduledTask -Action
$Action -Trigger $Trigger -Settings
(New-ScheduledTaskSettingsSet)
```

Finally, we'll actually create the scheduled task on the system, calling it File Transfer Automation and running it under the local administrator account with the provided password.

```
$Task | Register-ScheduledTask -TaskName
'File Transfer Automation' -User
'administrator' -Password 'supersecret'
```

This would register the script, and it will now copy all files from your source to the destination every day at 3 a.m.



Automate Data Encryption Using PowerShell

In today's world, it's crucial to protect a company's data. Encryption is a standard tool to do this. However, when a company has hundreds of thousands or millions of files, encrypting them can turn into a management nightmare. In this article, we'll cover how to automate this process.

In today's dangerous cyber environment, it's more important than ever to protect your data. Bad guys are always on the lookout for an easy score. As a sysadmin, it's one of your many jobs to set up security controls and make sure your network is not an easy target.

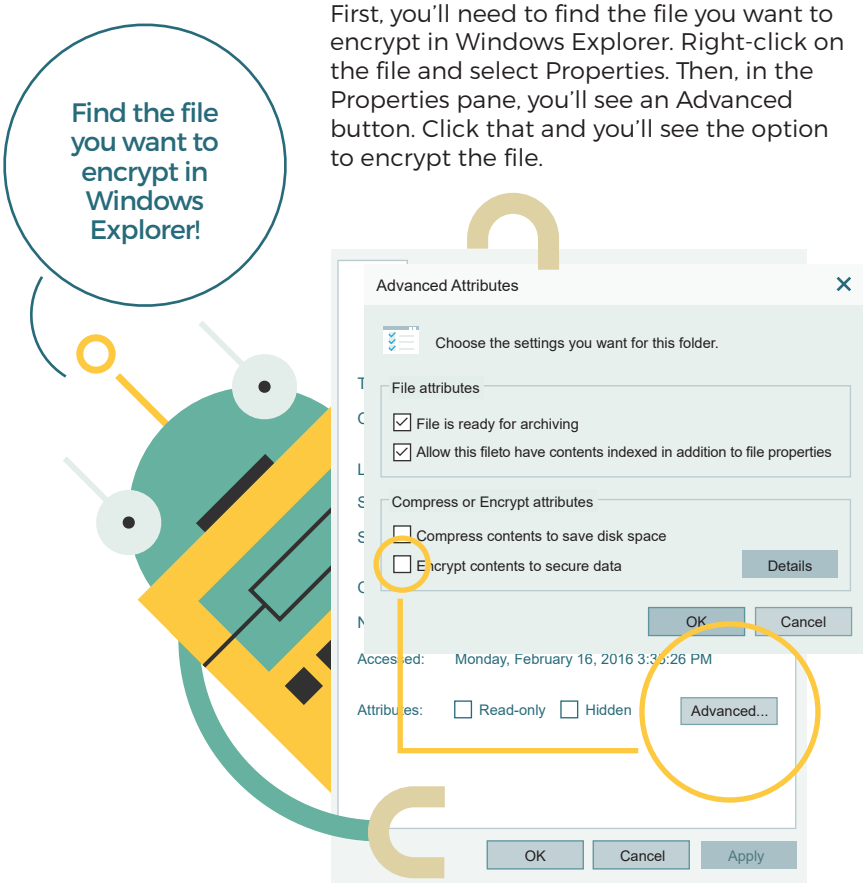
One way to do that is to ensure your network perimeter is secured to prevent any unauthorized access. However, what if your network is breached anyway? Perhaps someone physically comes into your data center and steals a server to gather valuable data you may have stored on it. If your data is not encrypted, kiss it goodbye. But, if you had the foresight to encrypt the data on that server beforehand, while your data might still be gone, at least you'll know it won't be read.

Encrypting data is always a good idea but it can be hard to manage, especially across different servers and storage locations. By using Microsoft's built-in Encrypting File System (EFS) technology and PowerShell, the task of encrypting and decrypting one, two or millions of files across your data center can be a lot easier.

In this article, I'll show you how you can manually encrypt and decrypt files with EFS using the GUI. Finally, I'll go over some PowerShell code that will allow you to perform this task over many different locations at once.

Encrypt Files via the GUI

First, you'll need to find the file you want to encrypt in Windows Explorer. Right-click on the file and select Properties. Then, in the Properties pane, you'll see an Advanced button. Click that and you'll see the option to encrypt the file.



Select the "Encrypt contents to secure data" checkbox and apply the change to immediately encrypt the file. You'll notice the file icon will change.

Automating Data Encryption

In a business environment, you're probably going to have to encrypt an entire folder or many different folders across different locations. If you'd rather not spend your time encrypting them manually, there's a better way: use PowerShell.

By using a PowerShell script, you can build code that will allow you to pass any number of files or folders into it to automatically encrypt them regardless of where they are.

Fortunately, Microsoft was kind to us and doesn't require a lot of scripting to make this happen. The act of encrypting and decrypting a file is as simple as calling an Encrypt() and Decrypt() method on a particular type of object, which can easily be obtained with Get-Item or, in the case of an entire folder(s), with Get-ChildItem.

For example, if I wanted to encrypt our example above with PowerShell, I'd only need a single line of code.

```
(Get-Item –Path C:\Groups.csv).Encrypt()
```

To decrypt:

```
(Get-Item –Path C:\Groups.csv).Decrypt()
```

Performing an encrypt or decrypt on an entire folder is just as easy. But, instead of using Get-Item, you'll need to use Get-ChildItem to get all of the files from within that folder.

```
(Get-ChildItem –Path
C:\Documents).Encrypt()
```

Using PowerShell Functions

I personally like using PowerShell functions and cmdlets instead of .NET methods such as Encrypt() and Decrypt(). So, I'm going to build "wrapper" functions that will allow me to use Enable-FileEncryption and Disable-FileEncryption instead. To help explain how this works, let's take a look at the script.

You can download an example script to test this out. To use this script, open up a PowerShell console and "dot source" the script into your current session.

```
. C:\EFS.ps1
```

This will bring in each function declared in the script. You can now use the functions to encrypt and decrypt any files you want. For example, to encrypt a file I can use Enable-FileEncryption.

```
Get-Item C:\Groups.csv |
Enable-FileEncryption
```

To decrypt, I can do the opposite.

```
Get-Item C:\Groups.csv |
Disable-FileEncryption
```

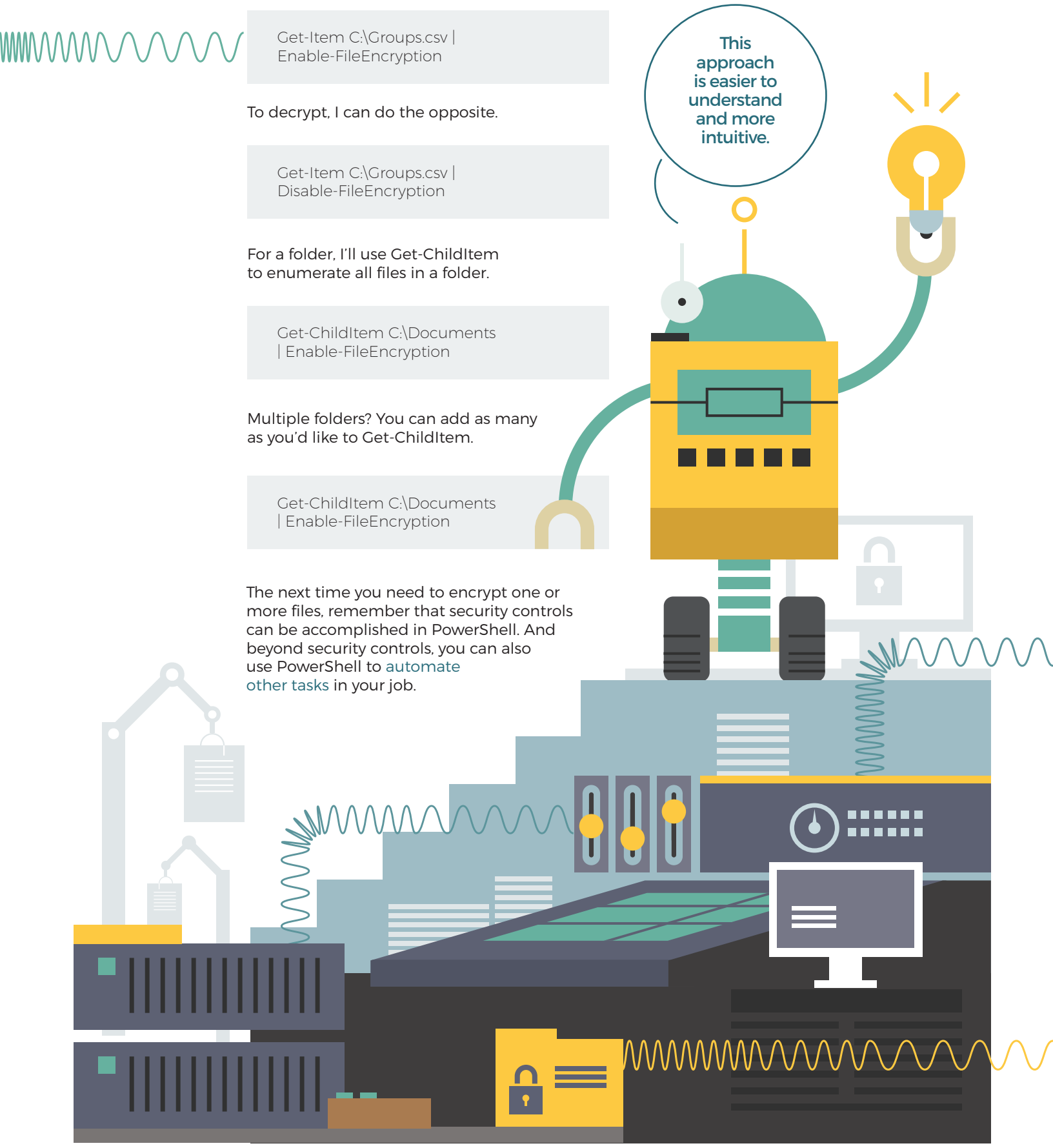
For a folder, I'll use Get-ChildItem to enumerate all files in a folder.

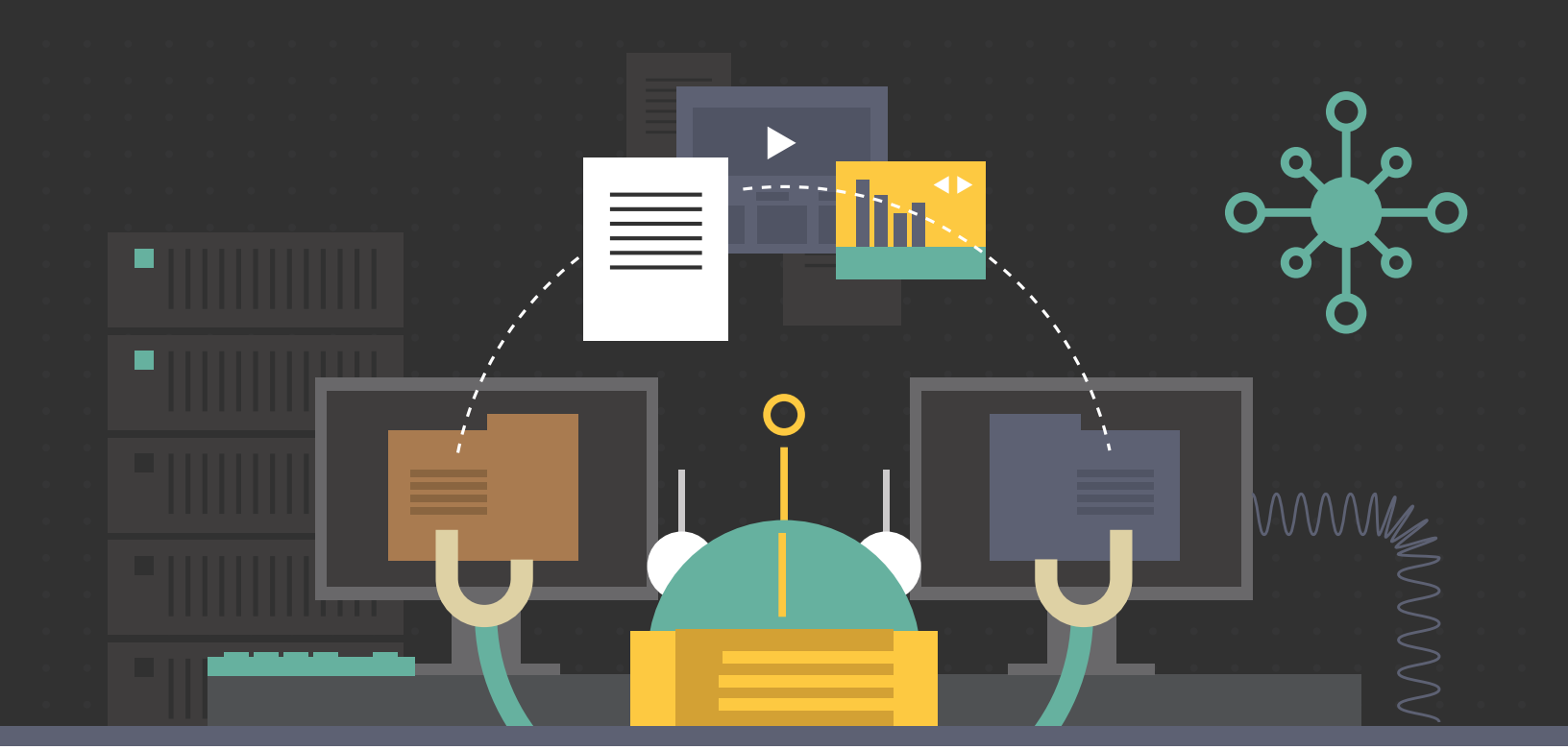
```
Get-ChildItem C:\Documents
| Enable-FileEncryption
```

Multiple folders? You can add as many as you'd like to Get-ChildItem.

```
Get-ChildItem C:\Documents
| Enable-FileEncryption
```

The next time you need to encrypt one or more files, remember that security controls can be accomplished in PowerShell. And beyond security controls, you can also use PowerShell to automate other tasks in your job.





How to Use PowerShell Copy-Item Cmdlet to Transfer Files Over WinRM

In complex environments, it's not always possible to transfer files the "traditional" way. Some companies have locked down DMZs or other environments that do not allow SMB file transfers but do allow servers to be remotely managed via Windows Remoting (WinRM). This article will show you how this can be taken advantage of by using this tunnel to transfer files.

It's easy to copy files with PowerShell Copy-Item via the command line. Once you specify the source and destination location, it just happens. Unfortunately, many administrators don't think about how this process occurs until it doesn't work. Whether or not you think about this, all TCP network communication (such as SMB file copies) use network ports to make the bits transfer. For a file copy process to get a file from point A to point B, a port needs to be open all the way to the destination node. In the case of an SMB file copy, that port is 445. This is a common port that's usually open internally, except in some high-security situations or across a DMZ.

PowerShell Copy-Item

If you're in a high-security environment or need to transfer files from an internal network to a DMZ that might have various port restrictions, how can you ensure that your scripts are able to copy files to nodes all the time? One way to

do so is to use PowerShell v5's Copy-Item cmdlet with the new –ToSession parameter.

This parameter was introduced with Windows Management Framework (WMF) v5 with the Copy-Item cmdlet. It provides a way to transfer files over the same link that you might use today to execute commands remotely on computers with cmdlets like Invoke-Command.

This process has a few different advantages, but included in the biggest benefits are the TCP ports used: 5985 (HTTP) and 5986 (HTTPS). These standard ports are typically open to manage remote nodes, sometimes even to a DMZ environment. By using Copy-Item –ToSession, an administrator can ensure files will always be copied regardless of whether or not SMB is blocked.

When you're using PowerShell Copy-Item via the traditional SMB method, you need to specify the Path and Destination parameters. If you'd like to copy a file called file1.txt inside of C:\Folder to a remote computer SERVER1 on its C:\, you could do this:

```
Copy-Item –Path C:\Folder1\file1.txt
–Destination '\\SERVER1\c$'
```

Notice that you're using the UNC path of \\SERVER1\c$ here. This will be important in a minute.

PowerShell Remoting Sessions

But what if SMB is blocked for some reason or you're using Invoke-Command to run commands on SERVER1 anyway? You can leverage PowerShell remoting sessions to transfer the file over WinRM instead of SMB. In order to do this, you must establish a new remoting session and then pass the file over that session.

First, you should create a new PowerShell remoting session. To do this, you can use the New-PSSession cmdlet and assign the session to the $session variable.

```
$session = New-PSSession
–ComputerName SERVER1
```

This will use Kerberos authentication to establish a new PowerShell remoting session, which is the most common method to use when in an Active Directory environment.

Next, you need to specify the ToSession parameter and a local path on the remote computer for the Destination parameter.

```
Copy-Item –Path C:\Folder1\file1.txt
–Destination 'C:\' –ToSession $session
```

Notice that you're now using C:\ for the destination rather than a UNC path. This command will accomplish the exact same thing as your previous one, but it will use the session to encapsulate the file and transfer it via WinRM.

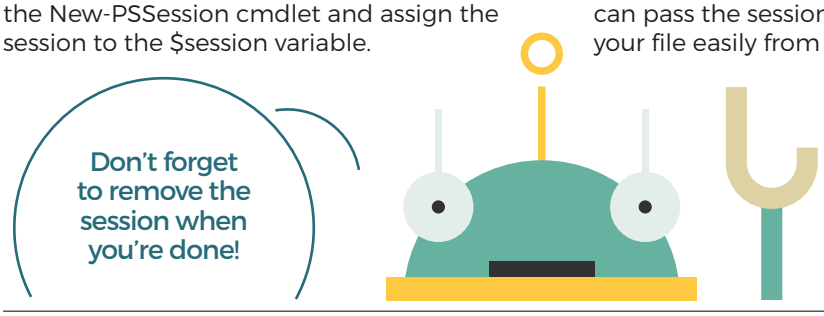
Don't forget to remove the session when you're done by using Remove-PSSession.

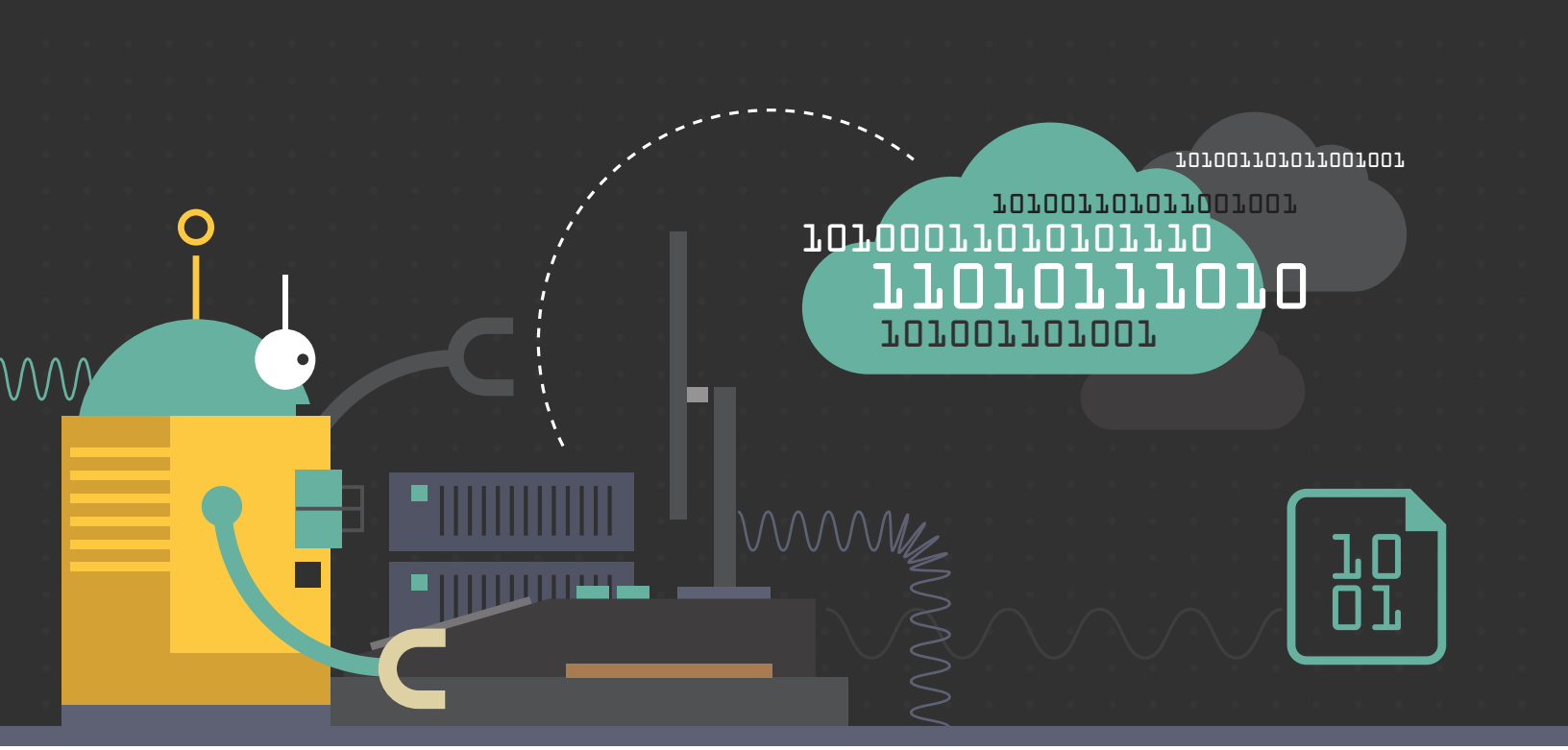
```
$session | Remove-PSSession
```

If you don't intend to reuse the session for anything else, you could also create the session and tear it down (all at the same time).

```
Copy-Item –Path C:\Folder1\file1.txt
–Destination 'C:\' –ToSession (New-
PSSession –ComputerName SERVER1)
```

That's all there is to it! The next time you find yourself in an environment where PowerShell remoting is allowed but SMB is restricted, or you're already using a remoting session for something else, you can pass the session to Copy-Item to get your file easily from point A to point B.





How to Copy Files Into a Microsoft Azure Storage Account

Cloud-first is a popular term nowadays. The cloud is slowly transforming IT. However, file management in the cloud is different than managing files on premise. In this article, we'll go over how you can transfers files stored on premise into Azure blob storage using PowerShell.

When working with Microsoft Azure, you'll inevitably come to a point to where you need to access material stored locally on premise. This might be a virtual disk in VHD format to use for Azure's IaaS service, a few PowerShell scripts you need executing on your Azure virtual machines or maybe just some configuration files for your Azure websites. Regardless, for your Azure resources to access these files, they'll need to be located in an Azure storage account.

The Set-AzureStorageBlob Content cmdlet

There are a couple of ways to transfer files stored locally into your Microsoft Azure storage account. I'll be doing this via the Set-AzureStorageBlobContent PowerShell cmdlet using the newer Azure Resource Manager (ARM) resources.

The Set-AzureStorageBlobContent is available in the Azure PowerShell module, so you'll need to ensure you get this module downloaded and available for use first. You'll also need an Azure subscription as well as a storage account to store your files. In this example, I'll assume you already have a storage container pre-created.

Once you meet these prerequisites, you can then use the Set-AzureStorageBlobContent cmdlet to transfer your local files and convert them into blob storage automatically.

Authenticate an Account

To get started you'll first need to authenticate your Azure subscription, which you can do using the Add-AzureRmAccount cmdlet. This will prompt you for a username and password, granting you the token necessary to make changes to your Azure subscription.

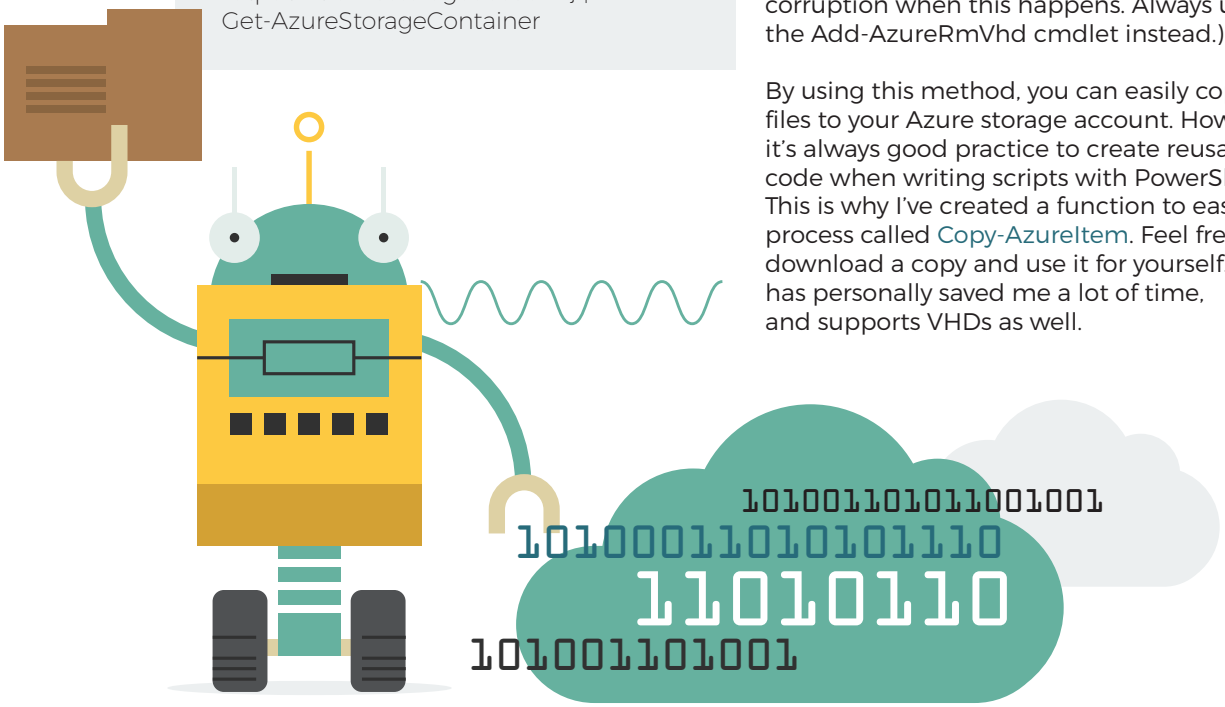
Once you've authenticated your Azure subscription, you'll need to specify a storage account in which to create your Azure storage blob. Your local files will automatically turn into blob storage once the file gets transferred to Azure. To specify a storage account, you can use the Get-AzureRmStorageAccount cmdlet. Below, I have two storage accounts available to me:

```
Get-AzureRmStorageAccount |
select storageaccountname
```

Now you need to specify a storage container inside of one of these storage accounts. You can do this by passing the storage account object directly to the Get-AzureStorageContainer cmdlet.

```
$storageContainer =
Get-AzureRmStorageAccount |
where {$_.StorageAccountName
-eq 'adbdemostorageaccount'} |
Get-AzureStorageContainer
```

By using this method, you can easily copy files to your Azure storage account.



Where's It Going?

You can see I've assigned this storage container to a variable, allowing me to quickly pass the object to the Set-AzureBlobStorageContent cmdlet. Once I have the storage container, I then need to define the local file path and the destination path. To do this, I'll use these all as parameters to the Set-AzureBlobStorageContent cmdlet.

```
$FilePath = 'C:\Users\Adam\MyFile.txt'
```

```
$BlobName = 'MyFile.txt'
```

```
$storageContainer |
Set-AzureStorageBlobContent –File
$FilePath –Blob $BlobName
```

You can see that I've defined a text file that was stored in the C:\Users\Adam folder and made the blob the same name as the file. But this is unnecessary. During the copy you can change the name, but I typically keep it the same name for simplicity.

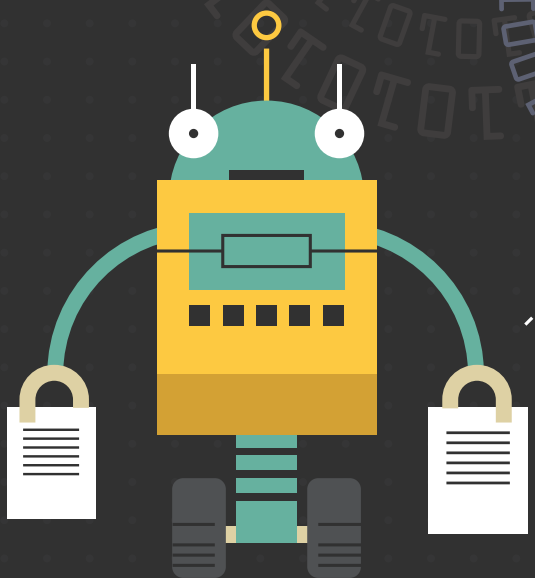
(Note: If you need to upload a VHD to an Azure storage account, do NOT use Set-AzureBlobContent. I've had issues with corruption when this happens. Always use the Add-AzureRmVhd cmdlet instead.)

By using this method, you can easily copy files to your Azure storage account. However, it's always good practice to create reusable code when writing scripts with PowerShell. This is why I've created a function to ease this process called Copy-AzureItem. Feel free to download a copy and use it for yourself. It has personally saved me a lot of time, and supports VHDs as well.

Conclusion

The applications of file transfer automation that were covered here only scratched the surface on the problem of file management a typical organization faces. Although using PowerShell is a powerful solution to the file management problem, it is still a language. That language can be complex and require a staff that understands code and can maintain it.

File transfer automation products like MOVEit from Ipswitch can implement the same level of automation without the need for you or your team to write code. However, if your team does have scripting expertise and would like to use that knowledge, you'll find that MOVEit can work in tandem with your scripts to build a robust file transfer automation engine.



Adam Bertram

Adam Bertram is an independent consultant, technical writer, trainer, and presenter. Adam specializes in consulting and evangelizing all things IT automation mainly focused around Windows PowerShell and Microsoft System Center. Adam is a Microsoft Windows Cloud and Datacenter Management MVP focused on Windows PowerShell and has numerous Microsoft IT pro certifications. He authors IT pro course content for Pluralsight, is 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.