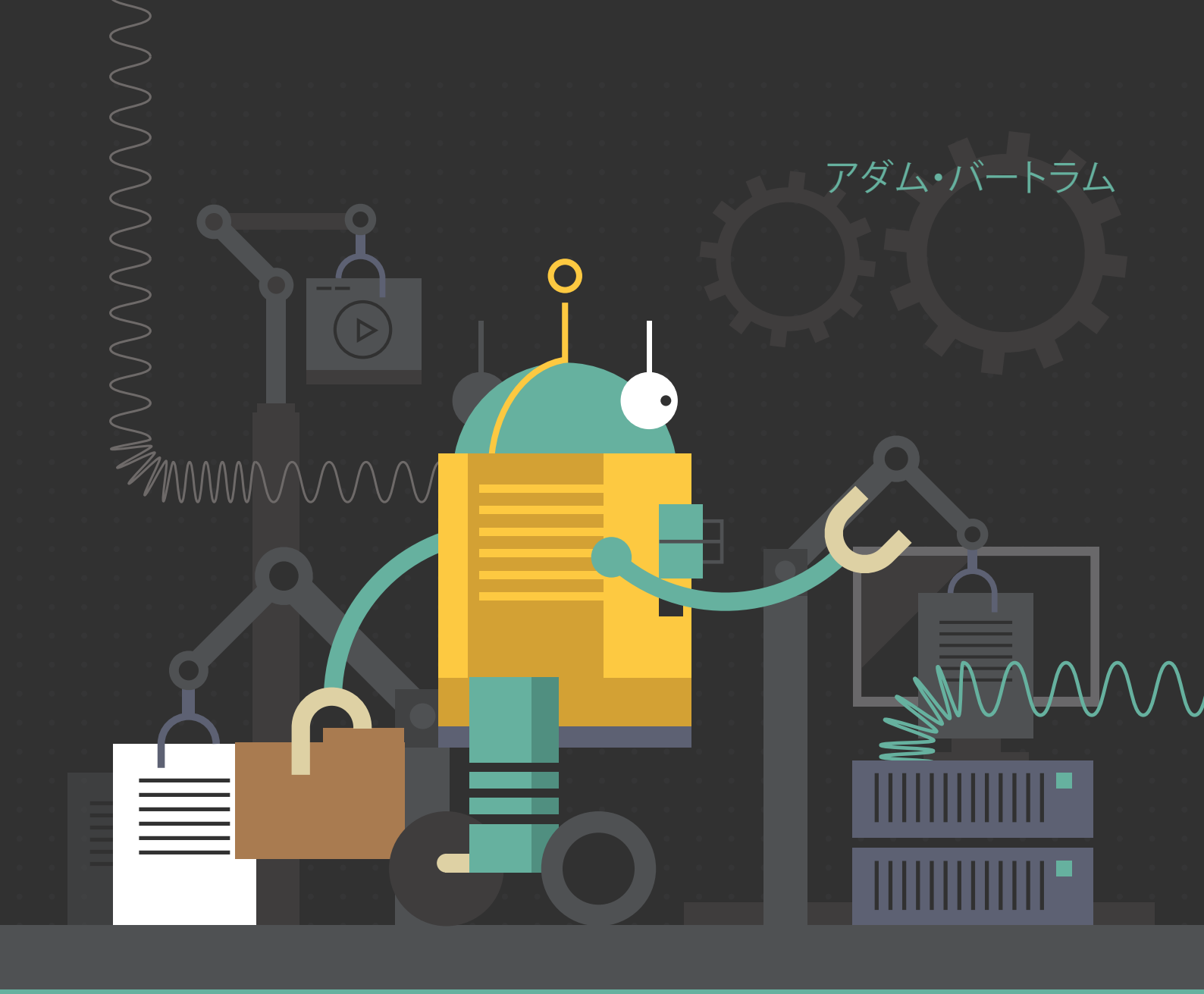
アダム・バートラム



PowerShell を使って自動化する方法

タスク、ファイル転送、
データ・セキュリティの自動化

ipswitch

はじめに

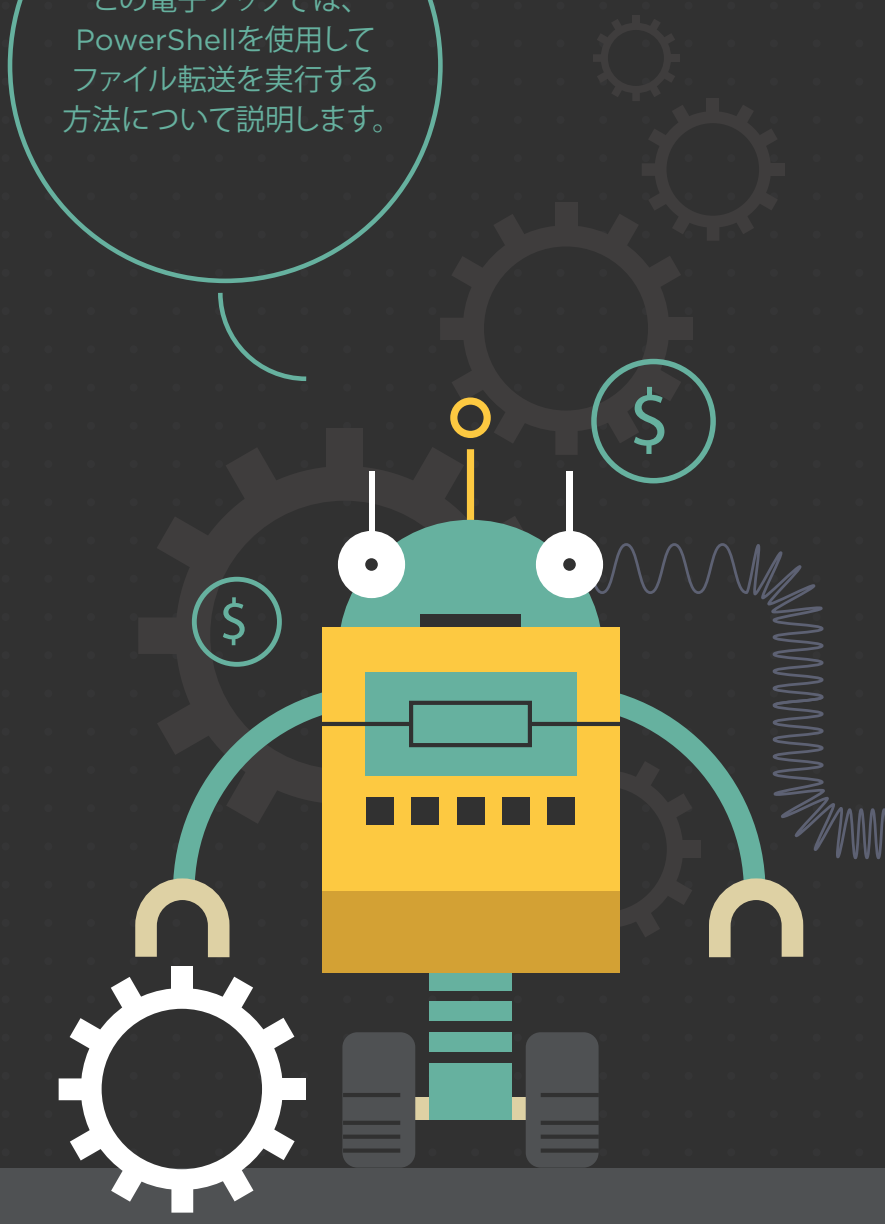
生産力の向上は、あらゆる組織にとって追求すべき目標です。しかし、予算に余裕がないまま、処理するべき仕事の量はますます増加しますから、必要なアウトプットを速やかに作成するのは大変な困難を伴います。IT部門はすでに人員削減されているところもあります。誤りを最小限に抑えて、予算内に収まるようにしながら、どのようにしてユーザーのニーズを満たしていけばいいでしょうか？答えは、自動化とPowerShellの使用にあります。

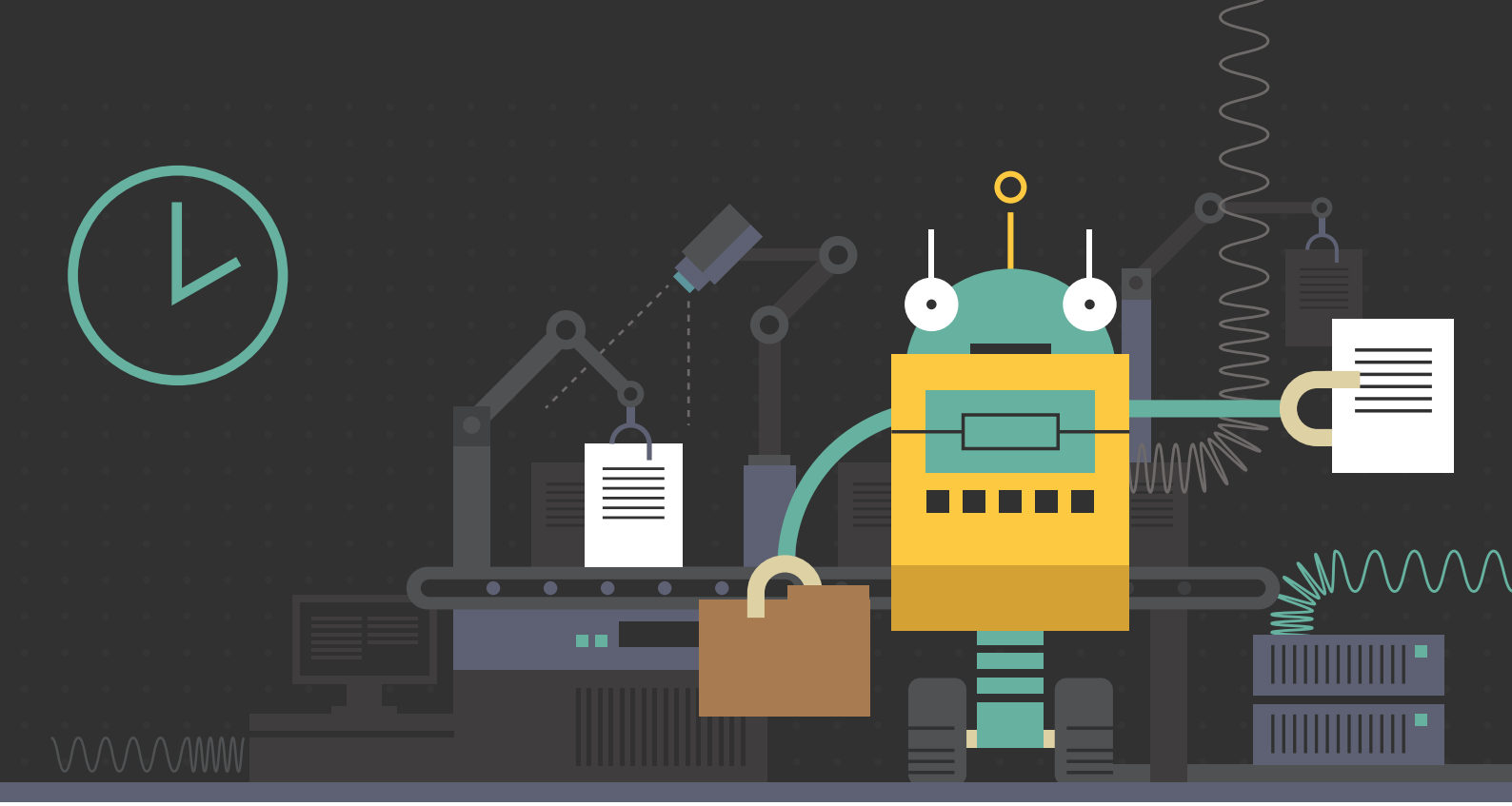
ITは、組織にとって単に便利なユーティリティではなく大切なビジネス資産になりつつあります。IT部門のメンバーがその能力を最大に活用できるエリア、問題をクリエイティブに解決すること、に集中できるよう、できる限り多くのプロセスを自動化することが不可欠です。

プロセスがルーチン化していて自動化に適しているタスクの１つに、ファイル転送があります。様々なデータはファイルがあり、IT部門は時には何百万ものファイルを処理することがあります。ファイルは、社内サーバーやクラウド内に保存され、外部組織に転送されたりすることもあります。すべてのアクティビティは継続的に発生します。ほとんどの場合、これらのファイル転送は予測可能です。ある部門では毎日特定の時間にレポートを必要とするかもしれません。パートナー会社が、新しい製品仕様を詳述した最新のExcelスプレッドシートを必要とするかもしれません。データベースをクラウド内でバックアップする必要があるかもしれません。

ファイル転送を自動化するには、どのような手法がいいでしょうか？1つの方法は、PowerShellを使って自分たちで自動化することです。必要に応じてどのファイル転送タスクでも自動化することができます。この電子ブックでは、IT部門のメンバーが、細々としたタスクに時間をとられることなく、問題解決や業務改善などの本来的に重要なタスクに時間をあてることができるようにするための手法をいくつか紹介します。

この電子ブックでは、PowerShellを使用してファイル転送を実行する方法について説明します。





スケジュールされたタスクをどのようにファイルトランスファーに適用できるか

ファイル転送には、転送を開始するトリガーがあります。そのトリガーは、ITスタッフによる何らかのアクションの実行でも、自動設定によるものでも、何でもかまいません。この章では、PowerShell を使用してファイル転送スクリプトを自動化するスケジュールされたタスクを作成する方法について説明します。

ファイルをある場所から別の場所にコピーすることは、どのように行っても簡単な作業です。Windowsエクスプローラでファイルをドラッグアンドドロップする方法、PowerShellで Copy-Itemを使う方法、DOSで単純なコピーコマンドを使用する方法など、さまざまな方法があります。ソースとデスティネーションのパスを指定し、その他のオプションのパラメータをいくつか設定するだけです。問題に遭遇するのは、多数のファイルを頻繁にコピーしなければならないときだけです。すべてのファイルがコピーされるのをその場にいてチェックする必要はありません。このような場合、スケジュールされたタスクを作成して自動化するのが最適です。

ファイルコピーを自動化する場合、特にWindows環境では、使用すべきスクリプト言語はWindows PowerShellになります。ある場所から別の場所に1つ以上のファイルを速やかにコピーする必要がある場合は、PowerShellを使用するのが便利です。PowerShellスクリプトを手動で開始するのは簡単ですが、Windowsのスケジュールされたタスクを使用してPowerShellスクリプト経由で転送を開始することもできます。

この章では、PowerShellを使用してファイル転送を実施する方法を説明します。スクリプトを作成し、定期的にそのスクリプトを始動させるスケジュールされたタスクを作成できます。ただし、この記述は、PowerShell v4がコンピュータにインストールされていることを前提としています。PowerShell v4がインストールされていない場合は、説明の通りに動作しない可能性があります。

スクリプトの作成

最初に、ファイル転送を実行するスクリプトを作成する必要があります。ここでは、CopyFiles.ps1という名前にしましょう。このスクリプトには次のコードが含まれます：

```
param(

[string]$SourcePath,

[string]$DestinationPath

)

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

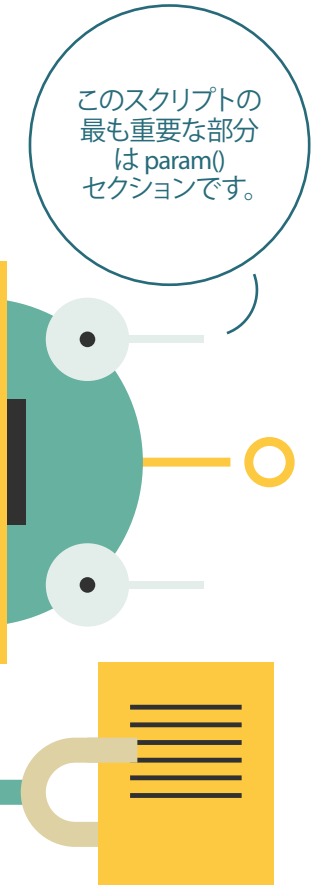
ご覧のように単純ですが、環境に応じて多くのカスタマイズが可能です。

このスクリプトの最も重要な部分は param() セクションです。これは、SourcePath と DestinationPath の2つのパラメータを含むパラメータブロックです。これらの両方をパラメータにすることで、スクリプトに異なる値を渡すことができ、再利用することができます。

SourcePath と DestinationPath がパラメータではなくて実際のパスである場合は、異なるファイルのコピーのたびに、別のスクリプトを作成する必要があります。このスクリプトを手動で実行する場合は、次のようになります。

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

この例では、C:\Sourceフォルダ内のすべてのファイルとサブフォルダを \\SERVER\Destination共有フォルダにコピーします。



スケジュールされたタスクの作成

CopyFiles.ps1 という PowerShellスクリプトができたので、これを実際にコンピュータが開始できるよう設定します。この例では、このスクリプトを1日1回、午前3時に実行するスケジュールされたタスクを作成することにします。

タスクスケジューラのGUIを実行してスケジュールされたタスクを作成することもできますが、自動化の観点から、PowerShellでスケジュールされたタスクを作成する方法を説明します。これを行うには、次の4つの手順を完了する必要があります。

1) スケジュールされたタスクアクションを作成。
2) トリガーを作成。
3) スケジュールされたタスクをメモリに作成。
4) スケジュールされたタスクをコンピュータ上に作成。

具体的には、次のようになります。まず、スケジュールされたタスクアクションを作成します。任意の引数とともに実行するEXEを定義します。ここでは、スクリプトがC:\CopyFiles.ps1 にあると仮定しています。

```
$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'"
```

次に、これを毎日午前3時に実行するようトリガーを作成します。

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

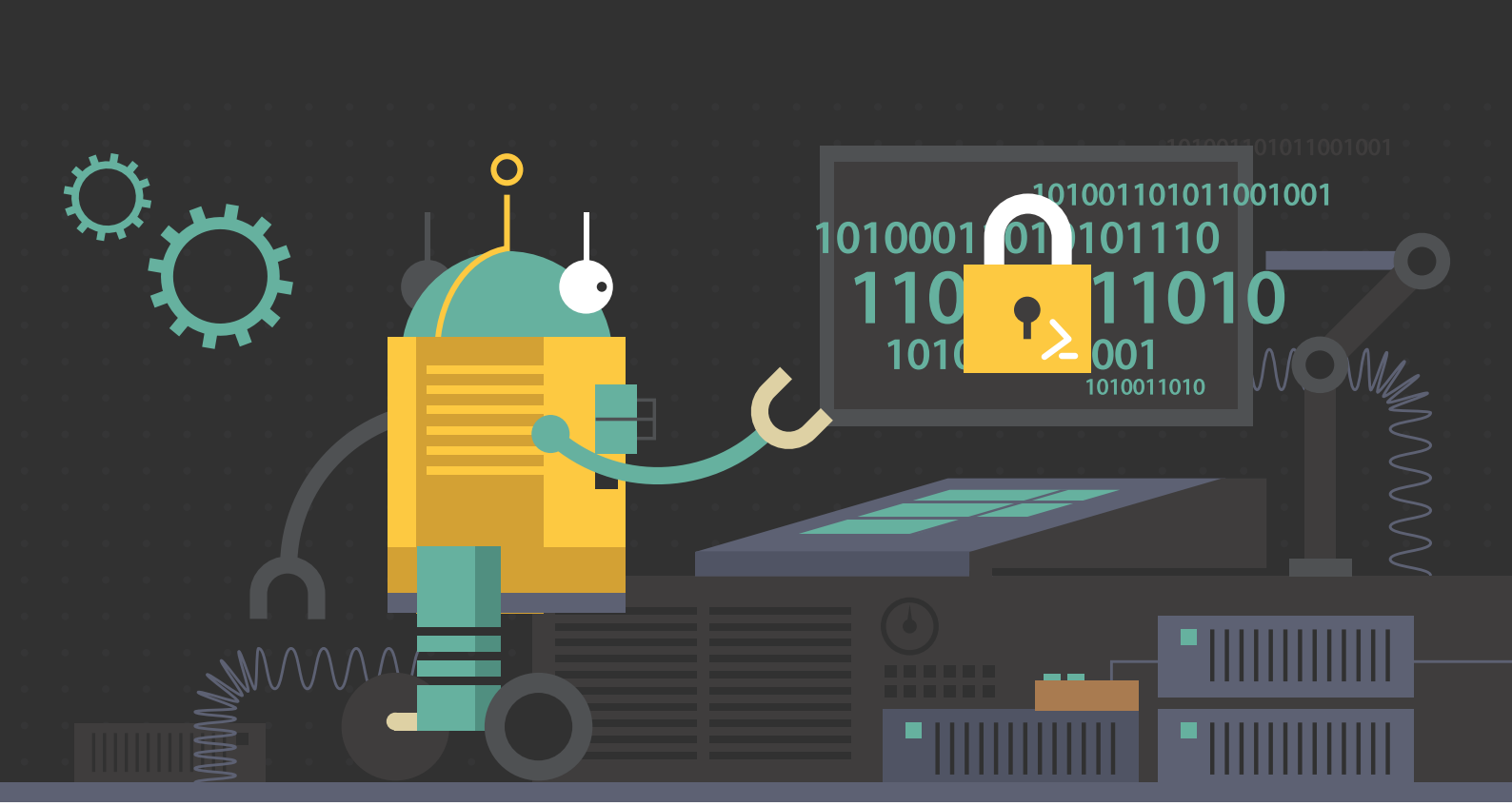
このあと、作成したアクションとトリガーを使ってスケジュールされたタスクをメモリに作成します。

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

後に、実際にシステム上にスケジュールされたタスクを作成します。タスク名は File Transfer Automation で、提供されたパスワードを使用するローカル管理者アカウントの下で実行されます。

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

これでスクリプトが登録され、毎日午前3時に、すべてのファイルがソースからデスティネーションにコピーされます。



PowerShellを使った データ暗号化の自動化

今日の世界では、企業が処理するデータを保護することは極めて重要です。データ保護のために通常利用される手段は暗号化です。しかし、数十万あるいは数百万ものファイルを処理しなければならない場合、それらを暗号化するよう管理するのは至難の技です。この章では、このプロセスを自動化する方法について説明します。

データ侵害の脅威がはびこる今日のサイバー環境においては、データを保護することはこれまで以上に重要です。データ侵害をたくらむ組織は、常に防御態勢が弱いところを探しています。システム管理者の多くの仕事のうちでも、セキュリティのコントロールをしっかり行い、管理しているネットワークが容易なターゲットにならないようにすることは、非常に大切です。

これを実行する1つの方法は、不正なアクセスを防ぐためにネットワーク境界を保護することです。しかし、保護対策をとっていたとしても、ネットワークに侵入されてしまう可能性は、ゼロにはなりません。誰かがデータセンターに物理的に侵入して、貴重なデータを収集するためにサーバーを盗んでしまうことも有り得ます。データが暗号化されていなければ、あきらめるしかありません。もし管理者に先見性があって、そのサーバー上のデータを事前に暗号化していたとすれば、データ損失は免れないとしても、少なくともデータを読み取られることはありません。

データの暗号化は常に良いアイデアですが、管理することは大変です。異なるサーバーがたくさん存在し、保管場所も多岐にわたれば、特に困難です。Microsoftの組み込み暗号化ファイルシステム（EFS）テクノロジーとPowerShellを使用すれば、１つでも２つでも、あるいは100万のファイルでも、データセンター全体のファイルの暗号化および復号化作業が非常に簡単になります。

この章では、GUIを使用してEFSでファイルを手動で暗号化および復号化する方法を示した後、この作業をさまざまな場所で一度に実行できるPowerShellコードについて説明します。

GUIを使用したファイルの暗号化

まず、Windowsエクスプローラで暗号化するファイルを探します。暗号化したいファイルが見つかったら、ファイル名の上で右クリックし、[Properties] を選択します。[Properties] ペインが表示されますので、右下の [Advanced...] ボタンをクリックしてください。[Advanced Attributes] ペインが表示されます。

[Encrypt contents to secure data] ボックスをクリックしてチェックマークを入れ、[OK] ボタンをクリックして変更を適用します。ファイルはすぐに暗号化され、ファイルアイコンが変わります。

データ暗号化の自動化

ビジネス環境では、おそらく、フォルダ全体または異なる場所にある多くの異なるフォルダを暗号化する必要があります。それらをすべて手動で暗号化するのは時間の無駄ですから、もっと効率的な方法を利用するべきです。PowerShellの使用を検討してください。

PowerShellスクリプトを使用すると、どこにあるファイルやフォルダでも、任意の数だけ自動的に暗号化することができるコードを作成することができます。

幸いにも、マイクロソフトのシステムはユーザーフレンドリーに設計されており、多くのスクリプトを書く必要はありません。ファイルの暗号化と復号化は、Get-Item で簡単に取得できる特定のオブジェクトに対して、あるいはフォルダ全体の場合はGet-ChildItemを使用して得られるオブジェクトに対して、Encrypt() または Decrypt() メソッドを呼び出すだけで実現できます。

たとえば、ここで用いている例をPowerShellで暗号化する場合は、1行のコードを書くだけです。

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

復号化のコードは次のようになります。

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

フォルダ全体を暗号化または復号化することも簡単です。フォルダ内のすべてのファイルを暗号化するには、Get-Item を使用する代わりに、Get-ChildItemを使用すればいいのです。

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

PowerShell 関数の利用

筆者は個人的には、Encrypt() やDecrypt() などの .NET メソッドよりも PowerShell関数とコマンドレットを使用する方法を好みます。上述した方法の代わりに、Enable-FileEncryption と Disable-FileEncryption を使用できるようにするラッパー関数を作成する方法を紹介します。この仕組みを理解するには実際のスクリプトを見るのがわかりやすいと思います。

ここをクリックするとサンプル・スクリプト*1付録をダウンロードできますので、テストしてみてください。このスクリプトを使用するには、PowerShellコンソールを開き、スクリプトを現在のセッションにドットソース形式で読み込みます。

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

そうすると、スクリプトで宣言された各関数を呼び出すことができます。これらの関数を使用して、どんなファイルでも暗号化や復号化ができます。たとえば、ファイルを暗号化するには、Enable-FileEncryption を使用できます。

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

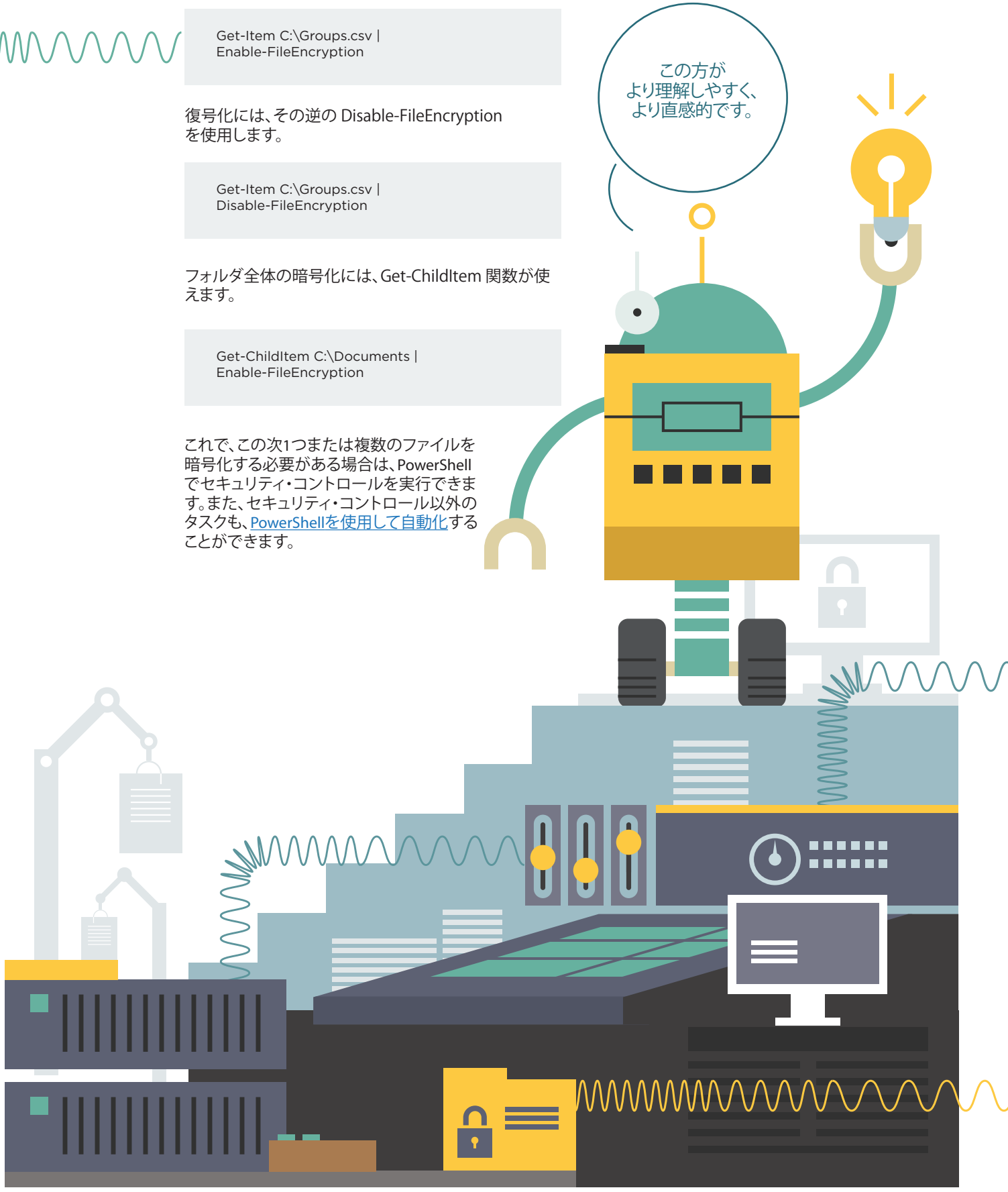
復号化には、その逆の Disable-FileEncryption を使用します。

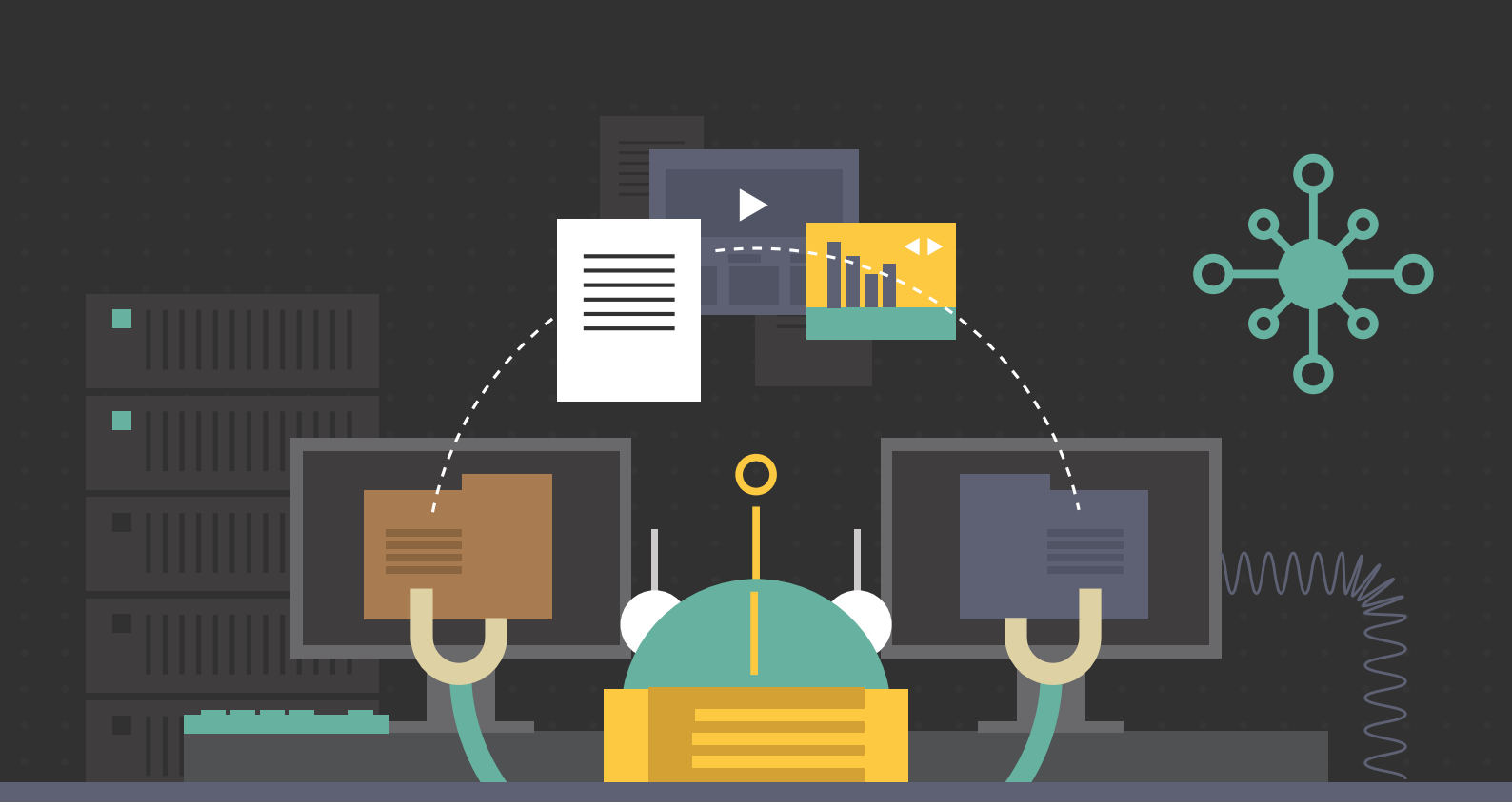
```
Get-Item C:\Groups.csv |
Disable-FileEncryption
```

フォルダ全体の暗号化には、Get-ChildItem 関数が使えます。

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

これで、この次1つまたは複数のファイルを暗号化する必要がある場合は、PowerShell でセキュリティ・コントロールを実行できます。また、セキュリティ・コントロール以外のタスクも、PowerShellを使用して自動化することができます。





PowerShell の Copy-Item コマンドレットを使って WinRM でファイル転送する方法

複雑なIT環境では、ファイルを「従来の」方法で転送することは必ずしも可能ではありません。DMZをロックダウンしたり、SMBファイル転送を許可しないが環境を、Windows Remote Management（WinRM）を使用してサーバーをリモート管理できるようにしている企業もあります。この章では、トンネルを使用してファイルを転送する方法を説明します

コマンドラインで PowerShell の Copy-Item を使ってファイルをコピーするのは簡単です。送信元と送信先の場所を指定するだけで実行できます。簡単なので、普段はこのプロセスの背後で何が起きているのか考えなくてもいいでしょうが、動作しなくなるとプロセスがどのように処理されるか理解する必要が出てきます。すべてのTCPネットワーク通信（SMBファイルコピーなど）は、ネットワークポートを使用してビットを転送します。ファイルコピー・プロセスでファイルをポイントAからポイントBに移動するには、宛先ノードまでポートが開いている必要があります。SMBファイルコピーの場合、そのポートは445です。これは、通常は内部に開いている一般的なポートですが、一部の高セキュリティ状況やDMZを通しての転送の場合はその限りではありません。

PowerShell の Copy-Item

セキュリティ・レベルが高い環境でファイル転送を行う場合や、内部ネットワークからさまざまなポート制限がある可能性のあるDMZにファイルを転送する必要がある場合に、スクリプトがいつでもファイルをノードにコピーできるようにするにはどうすればよいでしょうか？これを行う1つの方法は、PowerShell v5（2016年にリリース）のCopy-Itemコマンドレットを新しい -ToSession パラメータと共に使用することです。

この –ToSession パラメータは、Windows Management Framework（WMF）v5でCopy-Itemコマンドレットに使えるように導入されました。Invoke-Commandのようなコマンドレットを持つコンピュータ上でコマンドをリモートで実行して、現在使用しているのと同じリンクにファイルを転送することができます。

このプロセスにはいくつかの利点がありますが、最大のメリットは5985（HTTP）および5986（HTTPS）のTCPポートです。これらの標準ポートは、通常はリモートノードを管理するために開いていますが、DMZ環境にも適用されます。Copy-Item –ToSession を使用すれば、SMBがブロックされているかどうかにかかわらず、常にファイルが確実にコピーされるようにすることができます。

従来のSMBメソッドを使用して PowerShell Copy-Item を使用する場合は、Path と Destination のパラメータを指定する必要があります。C:\Folder1 の中にある file1.txt というファイルを、リモートコンピュータ SERVER1 の C:\ にコピーするには、次のようにします。

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

ここで、\\SERVER1\c$ というUNCパスを使用していることに注意してください。以下で説明します。

PowerShell の リモートセッション

しかし何らかの理由でSMBがブロックされている場合、またはInvoke-Commandを使用してSERVER1でコマンドを実行している場合はどうすればいいでしょうか？PowerShell のリモートセッションを利用して、SMB ではなく WinRM でファイルを転送することができます。これを行うには、新しいリモートセッションを確立し、そのセッションにファイルを渡します。

まず、新しいPowerShell リモートセッションを作成する必要があります。そのためには、New-PSSessionコマンドレットを使用し、$session変数にセッションを割り当てます。

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

これで、Kerberos認証を行って新しいPowerShellリモートセッションが確立されます。これは、Active Directory環境で使用される最も一般的な方法です。

次に、DestinationパラメータにToSessionパラメータとリモートコンピュータ上のローカルパスを指定する必要があります。

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

宛先に、UNCパスではなく、C:\ を使用することに注意してください。このコマンドは、以前のものとまったく同じことを実現しますが、セッションを使用してファイルをカプセル化し、WinRM経由で転送します。

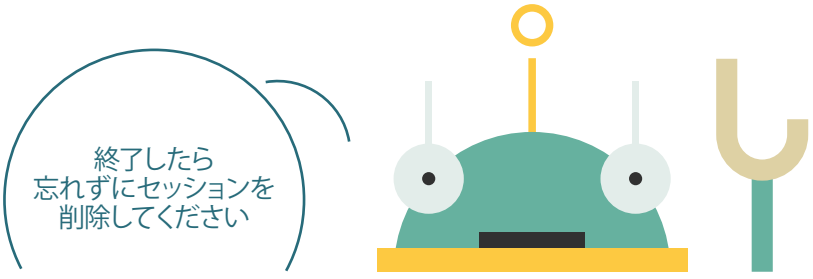
セッションが終了したら、Remove-PSSessionを使用してセッションを削除することを忘れないようにしてください。

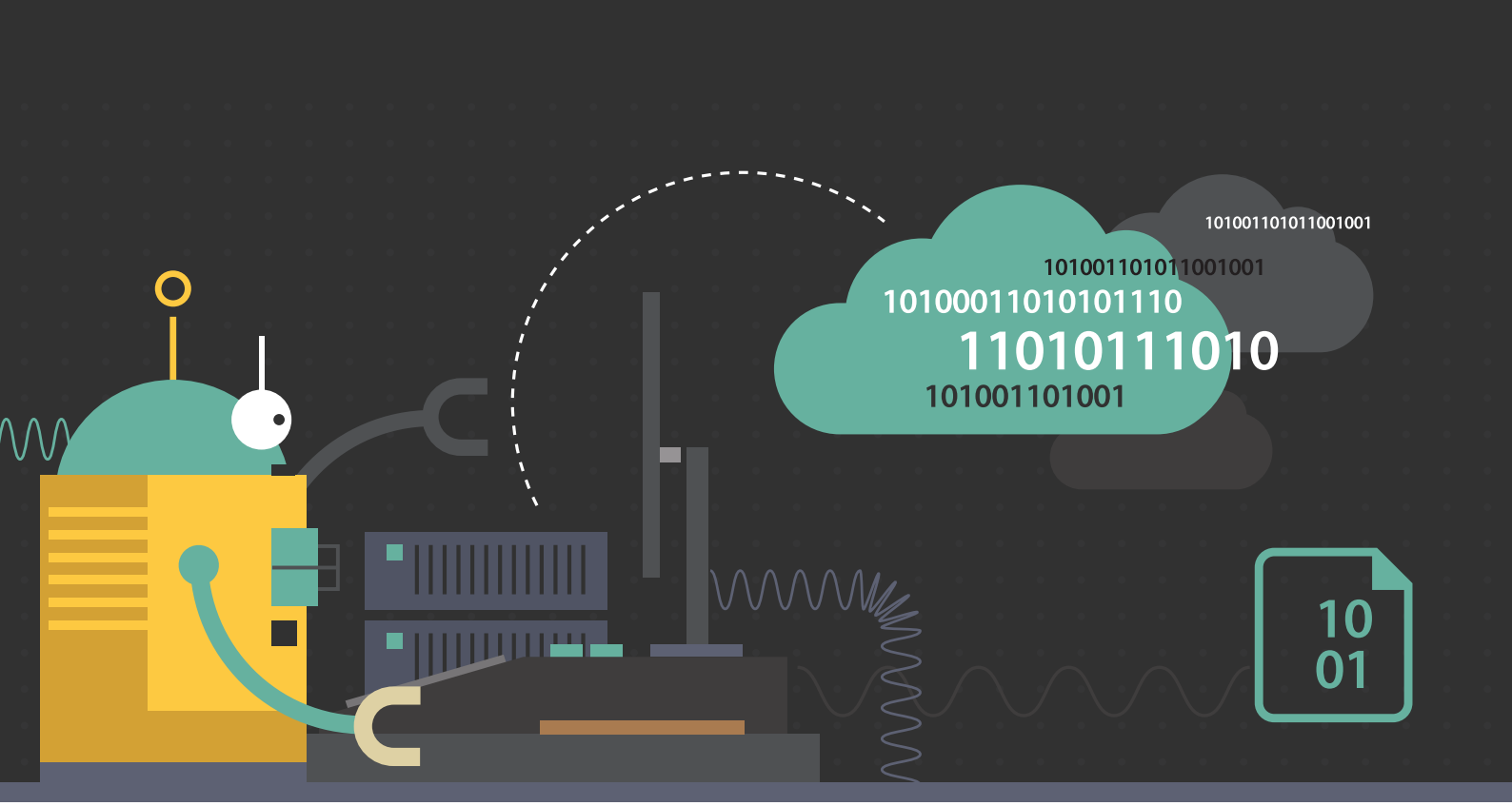
```
$session | Remove-PSSession
```

セッションを何か他の目的で再利用する必要がない場合は、セッションを作成してそれを破棄する一連の操作を一度に指定することもできます。

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

これですべてです！SMBが制限されていて、PowerShellのリモーティングが許可されている環境、またはすでに何か他の目的でリモーティング・セッションを使用している環境では、そのセッションをCopy-Itemにパスすることで、ファイルをポイントAからポイントBまで簡単に転送することができます。





ファイルをMicrosoft Azure ストレージアカウントに コピーする方法

クラウドファーストは今ではすっかり当たり前になりました。クラウドは徐々にITを変えています。ただし、クラウドでのファイル管理は、オンプレミス環境でのファイル管理とは異なります。この章では、PowerShellを使用してオンプレミスに保存されているファイルをAzure blob ストレージに転送する方法について説明します。

Microsoft Azureを使用する場合は、必然的に、オンプレミス環境にローカル保存されている資料にアクセスする必要があります。その資料は、AzureのIaaSサービスで使用するVHD形式の仮想ディスクや、Azure仮想マシンで実行する必要のあるPowerShellスクリプト、あるいはAzure Webサイトの設定ファイルだけかもしれません。Azureのリソースがこれらのファイルにアクセスするには、Azureストレージアカウント内に置かれている必要があります。

Set-AzureStorageBlobContent コマンドレット

ローカルに保存されたファイルをMicrosoft Azureストレージアカウントに転送するには、いくつかの方法があります。ここでは、新しいAzure Resource Manager（ARM）リソースを使用して、Set-AzureStorageBlobContent PowerShellコマンドレットでこれを実行する方法を説明します。

Set-AzureStorageBlobContentはAzure PowerShellモジュールで使用されるので、最初にこのモジュールをダウンロードして使用できるようにする必要があります。ファイルを保存するにはAzureサブスクリプションとストレージアカウントが必要です。ここは、ストレージコンテナが作成済みであると仮定します。

これらの前提条件が満たされれば、Set-AzureStorageBlobContent コマンドレットを使用してローカル・ファイルを転送し、それらを自動的に blob ストレージに変換することができます。

アカウントの認証

まずAzureサブスクリプションを認証する必要があります。これはAdd-AzureRmAccountコマンドレットを使用して行うことができます。ユーザー名とパスワードを入力し、Azureサブスクリプションを変更するために必要なトークンを獲得します。

Azureサブスクリプションを認証したら、Azure blobストレージを作成するストレージアカウントを指定する必要があります。ファイルがAzureに転送されると、ローカル・ファイルは自動的にblobストレージに格納されます。ストレージアカウントを指定するには、Get-AzureRmStorageAccount コマンドレットを使用します。次のようにすれば、2つのストレージアカウントが利用可能になります。

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

これらのストレージアカウントの1つの内部にストレージコンテナを指定する必要があります。これを行うには、Get-AzureStorageContainerコマンドレットにストレージアカウント・オブジェクトを直接渡します。

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

Azure ストレージアカウントへのファイル・コピー

変数にストレージコンテナを割り当てたので、すぐにオブジェクトをSet-AzureBlobStorageContentコマンドレットに渡すことができます。ストレージコンテナを取得したら、ローカルファイルパスと宛先パスを定義する必要があります。これを行うのに、これらすべてをSet-AzureBlobStorageContentコマンドレットのパラメータとして設定します。

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

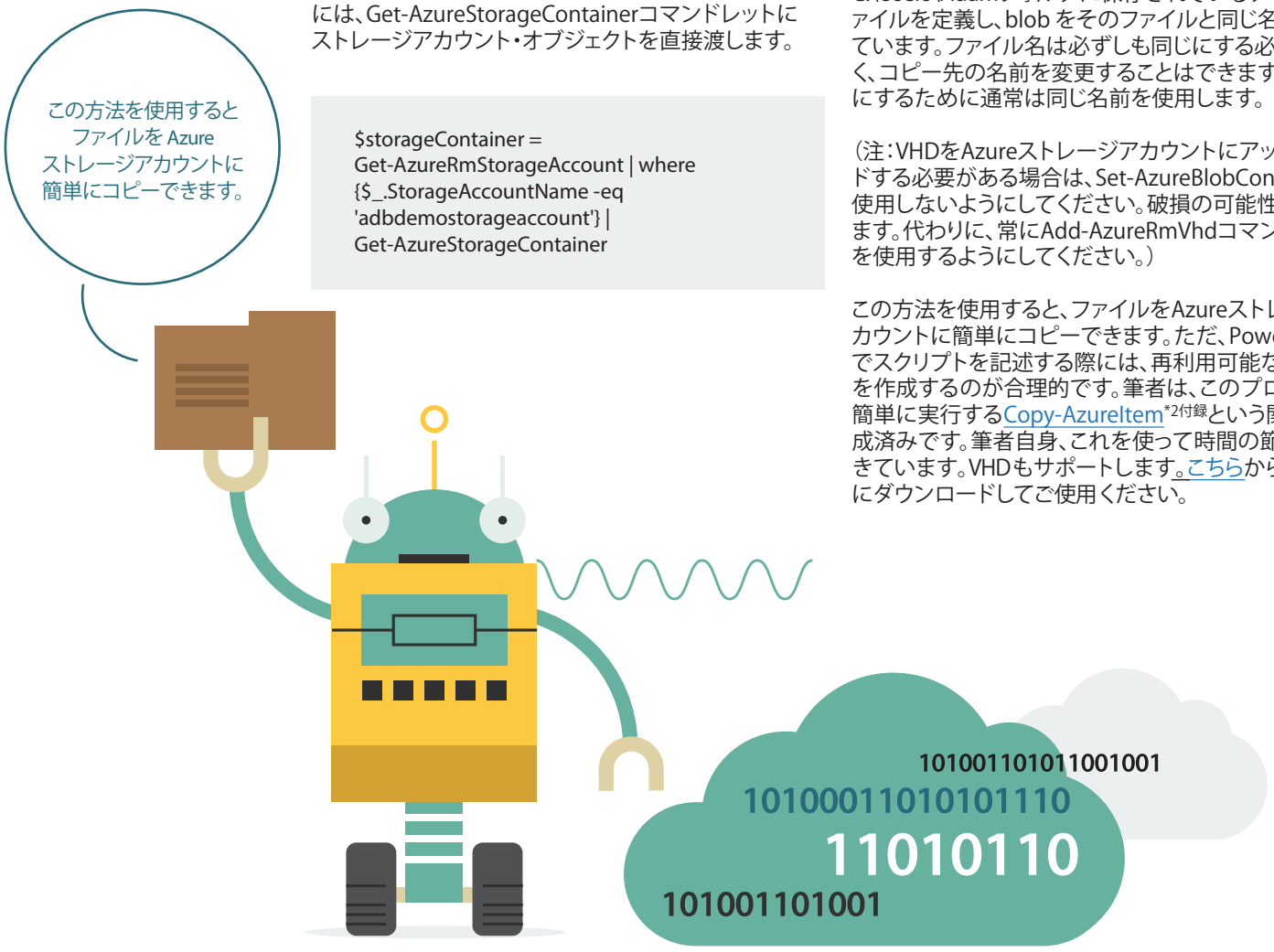
```
$BlobName = 'MyFile.txt'
```

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

C:\Users\Adamフォルダに保存されているテキストファイルを定義し、blob をそのファイルと同じ名前にしています。ファイル名は必ずしも同じにする必要はなく、コピー先の名前を変更することはできますが、簡単にするために通常は同じ名前を使用します。

（注：VHDをAzureストレージアカウントにアップロードする必要がある場合は、Set-AzureBlobContent は使用しないようにしてください。破損の可能性があります。代わりに、常にAdd-AzureRmVhdコマンドレットを使用するようにしてください。）

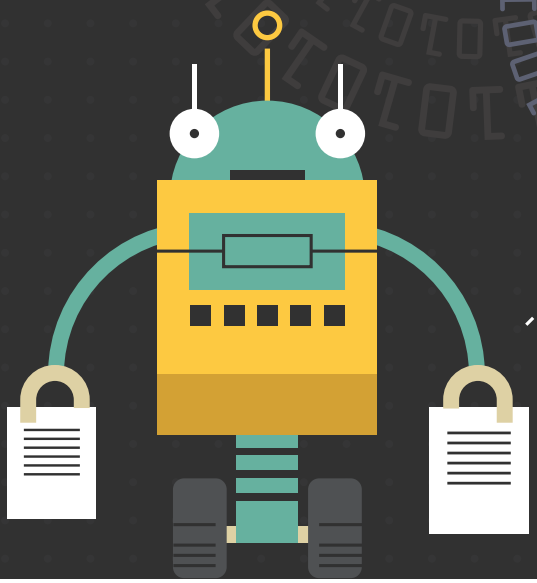
この方法を使用すると、ファイルをAzureストレージアカウントに簡単にコピーできます。ただ、PowerShellでスクリプトを記述する際には、再利用可能なコードを作成するのが合理的です。筆者は、このプロセスを簡単に実行するCopy-AzureItem[*2付録]という関数を作成済みです。筆者自身、これを使って時間の節約ができています。VHDもサポートします。こちらから、自由にダウンロードしてご使用ください。



おわりに

ここで取り上げたファイル転送自動化のアプリケーションは、たいていの組織が直面するファイル管理問題の表層的な部分だけに対するものです。PowerShellを使用することは、ファイル管理問題への強力なソリューションになりますが、PowerShellはスクリプト言語です。スクリプト言語は複雑なので、コードを理解し、維持できるスタッフが必要です。

イプスイッチのMOVEitのようなファイル転送自動化製品は、IT部門でコードを書かなくても、同じレベルの自動化を実装できます。もちろん、IT部門にスクリプトの専門知識があって、その知識を使用したい場合も、MOVEitがスクリプトと連携して堅牢なファイル転送自動化エンジンを構築できます。



著者:アダム・バートラン

アダム・バートランは独立したコンサルタント、テクニカルライター、トレーナー、プレゼンターであり、Windows PowerShellとMicrosoft System Centerを中心にIT自動化に関するコンサルティングと普及活動を行っています。アダムは、Microsoft Windows Cloud と Datacenter Management における Windows PowerShell のMVP（最重要人物）であり、多数のMicrosoft ITプロ認定を取得しています。PluralsightのITプロコースのコンテンツを作成し、さまざまなユーザーグループや会議で多数の印刷物やオンライン出版物に定期的に寄稿しています。アダムへのアクセスは、adamtheautomator.comまたは @adbertram の Twitterで可能です。

付録: 本文中のサンプルは、以下からもご覧いただけます。

***1: サンプル・スクリプト**

```
function Enable-FileEncryption
{
	<#
	.SYNOPSIS
		This function enables EFS file encryption on a file.

	.EXAMPLE
		PS> Get-Item -Path 'C:\File.txt' | Enable-FileEncryption

		This example finds the C:\File.txt with Get-Item, passes it through
the pipeline to Enable-FileEncryption which will
		then EFS encrypt the file.

	.EXAMPLE
		PS> Get-ChildItem -Path 'C:\Folder' | Enable-FileEncryption

		This example will encrypt every folder in C:\Folder.

	.PARAMETER File
		A System.IO.FileInfo object that will be encrypted.
	#>
	[OutputType([void])]
	[CmdletBinding()]
	param
	(
		[Parameter(Mandatory,ValueFromPipeline)]
		[ValidateNotNullOrEmpty()]
		[System.IO.FileInfo]$File
	)
	begin {
		$ErrorActionPreference = 'Stop'
	}
	process {
		try

		{
			$File.Encrypt()
		}
		catch
		{
			$PSCmdlet.ThrowTerminatingError($_)
		}
	}
}
```

```powershell
function Disable-FileEncryption
{
	<#
	.SYNOPSIS
		This function disables EFS file encryption on a file.

	.EXAMPLE
		PS> Get-Item -Path 'C:\File.txt' | Disable-FileEncryption

		This example finds the C:\File.txt with Get-Item, passes it through
the pipeline to Disable-FileEncryption which will
		then EFS decrypt the file.

	.EXAMPLE
		PS> Get-ChildItem -Path 'C:\Folder' | Disable-FileEncryption

		This example will decrypt every folder in C:\Folder.

	.PARAMETER File
		A System.IO.FileInfo object that will be decrypted.
	#>
	[OutputType([void])]
	[CmdletBinding()]
	param
	(
		[Parameter(Mandatory, ValueFromPipeline)]
		[ValidateNotNullOrEmpty()]
		[System.IO.FileInfo]$File
	)
	begin {
		$ErrorActionPreference = 'Stop'
	}

	process {
		try
		{
			$File.Decrypt()
		}
		catch
		{
			$PSCmdlet.ThrowTerminatingError($_)
		}
	}
}
```

***2: Copy-AzureItemコード**

```
function
Copy-
AzureItem
        {
                <#
                .SYNOPSIS
                        This function simplifies the process of uploading files to an
Azure storage account. In order for this function to work you
                        must have already logged into your Azure subscription with
Login-AzureAccount. The file uploaded will be called the file
                        name as the storage blob.

                .PARAMETER FilePath
                        The local path of the file(s) you'd like to upload to an Azure
storage account container.

                .PARAMETER ContainerName
                        The name of the Azure storage account container the file will be
placed in.

                .PARAMETER DestinationName
                        The name of the file stored as an Azure blob. By default, it
will be the same name as as the local file. Use this parameter
                        to give it a different name once uploaded.


                .PARAMETER BlobType
                        The type of blob you'd like the file to become when it gets
uploaded to the storage account. For example, when
                        uploading VHDs, you should only use Page.

                .PARAMETER ResourceGroupName
                        The name of the resource group the storage account is in.

                .PARAMETER StorageAccountName
                        The name of the storage account the container that will hold the
file is in.
                #>
                [CmdletBinding()]
                param
                (
                        [Parameter(Mandatory, ValueFromPipelineByPropertyName)]
                        [ValidateNotNullOrEmpty()]
                        [ValidateScript({ Test-Path -Path $_ -PathType Leaf })]
                        [Alias('FullName')]
                        [string]$FilePath,

                        [Parameter(Mandatory)]
                        [ValidateNotNullOrEmpty()]
                        [string]$ContainerName,
```

```
        [Parameter(Mandatory)]
        [ValidateNotNullOrEmpty()]
        [string]$ResourceGroupName,

        [Parameter(Mandatory)]
        [ValidateNotNullOrEmpty()]
        [string]$StorageAccountName,

        [Parameter()]
        [ValidateNotNullOrEmpty()]
        [string]$DestinationName,

        [Parameter()]
        [ValidateNotNullOrEmpty()]
        [ValidateSet('Page', 'Block')]
        [string]$BlobType = 'Page'
)
begin

        {
                $ErrorActionPreference = 'Stop'
        }
        process
        {
                try
                {
                        $saParams = @{
                                'ResourceGroupName' = $ResourceGroupName
                                'Name' = $StorageAccountName
                        }

                        $storageContainer = Get-AzureRmStorageAccount @saParams |
Get-AzureStorageContainer -Container $ContainerName

                        if (-not
$PSBoundParameters.ContainsKey('DestinationName'))
                        {
                                $DestinationName = $FilePath | Split-Path -Leaf
                        }

                        ## Use Add-AzureRmVhd if the file is a VHD. Set-
AzureStorageBlobContent is known to corrupt the large VHD when uploading
                        if ($FilePath.EndsWith('.vhd'))
                        {
                                $destination = ('{0}{1}/{2}' -f
$storageContainer.Context.BlobEndPoint, $ContainerName, $DestinationName)
                                $vhdParams = @{
                                        'ResourceGroupName' = $ResourceGroupName
                                        'Destination' = $destination
                                        'LocalFilePath' = $FilePath
                                }
```

```
                        Write-Verbose -Message "Uploading
[$($vhdParams.LocalFilePath)] to [$($vhdParams.Destination)] in resource group
[$($vhdParams.ResourceGroupName)]..."
                        Add-AzureRmVhd @vhdParams
                    }
                    else
                    {
                        $bcParams = @{
                            'File' = $FilePath
                            'BlobType' = $BlobType
                            'Blob' = $DestinationName

                        }
                        $storageContainer | Set-AzureStorageBlobContent
@bcParams
                    }
                }
                catch
                {
                    Write-Error -Message $_.Exception.Message
                }
            }
}
```