* 10 minutes

Complex or repetitive tasks often take a great deal of administrative time. Organizations prefer to automate these tasks to reduce costs and avoid errors.

This is important in the Customer Relationship Management (CRM) company example. There, you’re testing your software on multiple Linux Virtual Machines (VMs) that you need to continuously delete and recreate. You want to use a PowerShell script to automate the creation of the VMs versus creating them manually each time.

Beyond the core operation of creating a VM, you have a few additional requirements for your script:

* You will create multiple VMs, so you want to put the creation inside a loop
* You need to create VMs in three different resource groups, so the name of the resource group should be passed to the script as a parameter

In this section, you will see how to write and execute an Azure PowerShell script that meets these requirements.

## What is a PowerShell script?

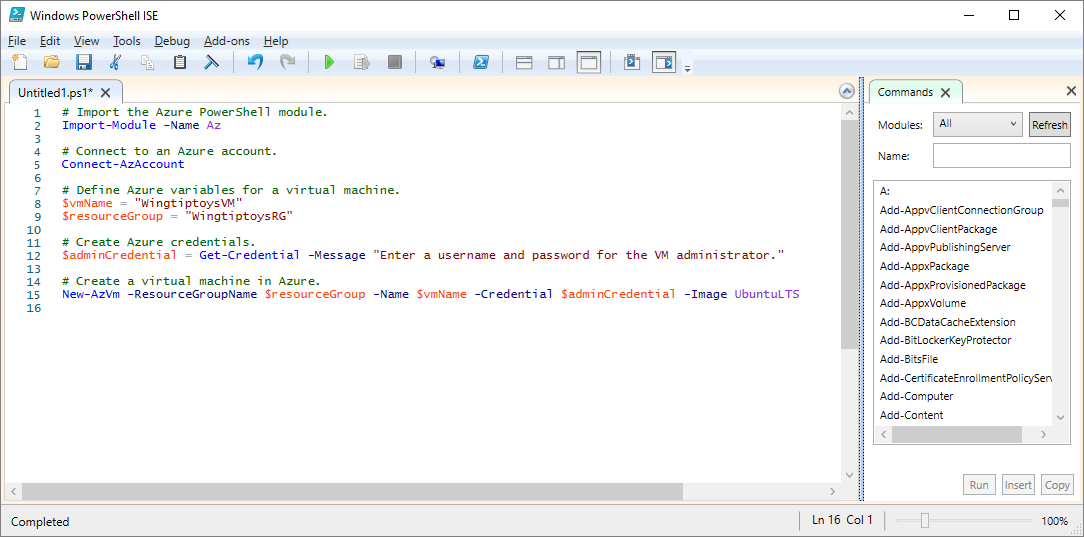
A PowerShell script is a text file containing commands and control constructs. The commands are invocations of cmdlets. The control constructs are programming features like loops, variables, parameters, comments, etc., supplied by PowerShell.

PowerShell script files have a **.ps1** file extension. You can create and save these files with any text editor.

Tip

If you’re writing PowerShell scripts under Windows, you can use the Windows PowerShell Integrated Scripting Environment (ISE). This editor provides features such as syntax coloring and a list of available cmdlets.

The following screenshot shows the Windows PowerShell Integrated Scripting Environment (ISE) with a sample script to connect to Azure and create a virtual machine in Azure.



Screenshot of the Windows PowerShell Integrated Scripting Environment with a script to create a virtual machine open in the editing window.

Once you’ve written the script, execute it from the PowerShell command line by passing the name of the file preceded by a dot and a backslash:

.\myScript.ps1

## PowerShell techniques

PowerShell has many features found in typical programming languages. You can define variables, use branches and loops, capture command-line parameters, write functions, add comments, and so on. We will need three features for our script: variables, loops, and parameters.

### Variables

As you saw in the last unit, PowerShell supports variables. Use **$** to declare a variable and **=** to assign a value. For example:

$loc = "East US"  
$iterations = 3

Variables can hold objects. For example, the following definition sets the **adminCredential** variable to the object returned by the **Get-Credential** cmdlet.

$adminCredential = Get-Credential

To obtain the value stored in a variable, use the **$** prefix and its name, as in the following:

$loc = "East US"  
New-AzResourceGroup -Name "MyResourceGroup" -Location $loc

### Loops

PowerShell has several loops: **For**, **Do…While**, **For…Each**, and so on. The **For** loop is the best match for our needs, because we will execute a cmdlet a fixed number of times.

The core syntax is shown below; the example runs for two iterations and prints the value of **i** each time. The comparison operators are written **-lt** for “less than”, **-le** for “less than or equal”, **-eq** for “equal”, **-ne** for “not equal”, etc.

For ($i = 1; $i -lt 3; $i++)  
{  
 $i  
}

### Parameters

When you execute a script, you can pass arguments on the command line. You can provide names for each parameter to help the script extract the values. For example:

.\setupEnvironment.ps1 -size 5 -location "East US"

Inside the script, you’ll capture the values into variables. In this example, the parameters are matched by name:

param([string]$location, [int]$size)

You can omit the names from the command line. For example:

.\setupEnvironment.ps1 5 "East US"

Inside the script, you’ll rely on position for matching when the parameters are unnamed:

param([int]$size, [string]$location)

We could take these parameters as input and use a loop to create a set of VMs from the given parameters. We’ll try that next.

The combination of PowerShell and Azure PowerShell gives you all the tools you need to automate Azure. In our CRM example, we’ll be able to create multiple Linux VMs using a parameter to keep the script generic and a loop to avoid repeated code. This means that we can execute a formerly complex operation in a single step.

## Next unit: Exercise - Create and save scripts in Azure PowerShell

[Continue](https://docs.microsoft.com/en-us/learn/modules/automate-azure-tasks-with-powershell/8-exercise-create-resource-using-script/)