# Auto-Start Virtual Machine Part 2

In this tutorial I will be creating a PowerShell script to automatically start an existing VirtualBox virtual machine that has Guest Additions installed. After starting the guest VM, we will check whether the SSHD service is running on the guest VM. Once confirmed, a PuTTY passwordless SSH connection will be launched to provide quick access to the VM. Finally, each step in the process will be logged.

During the tutorial, I will be using VBoxManage guestproperty which allows you to get and set properties of a running VM. I will also be using VBoxManage guestcontrol which enables control of the VM from the host machine. The use of both **VBoxManage guestproperty** and **guestcontrol** requires that Guest Additions be installed on the VM.

Before proceeding, please note that in order to execute the lone call to **VBoxManage guestcontrol** on my Windows 10 PC (also tested on my Windows 7 SP1 laptop), I had to install GitBash. PowerShell has many aliased cmdlets such as **cd** -> **Set-Location**, **pwd** -> **Get-Location** and many more. **GitBash** will allow us to run Unix/Linux commands that haven't been aliased to PowerShell cmdlets, such as **grep** and **wc**.

Refer to the prerequisites listed below to complete this tutorial.

## Prerequisites

- VirtualBox VM with Guest Additions installed
- PuTTY client for managing SSH sessions
- SSH keypair
- GitBash installed

For instructions on how to install VirtualBox and extension pack, see my **VirtualBox Install** tutorial **here**.
If you do not already have a virtual machine, my other tutorial, **CentOS 7 Server Install**, can be accessed **here**.
My **Guest Additions Install** tutorial can be accessed **here**. My **PuTTY Passwordless SSH** tutorial is **here**.
Finally, my **GitBash Install** tutorial can be accessed **here**.

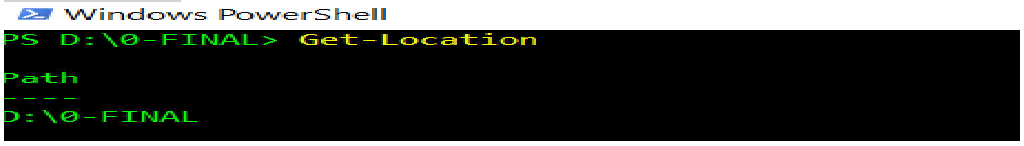
### Steps to complete tutorial:

- Create Script
- Review Script
- Execute Script

## Create Script

I have created an empty script file named **auto_start_vm_2.ps1**.
First, we will set the destination of the logfile. To do this, we will first get the present working directory with **Get-Location**:

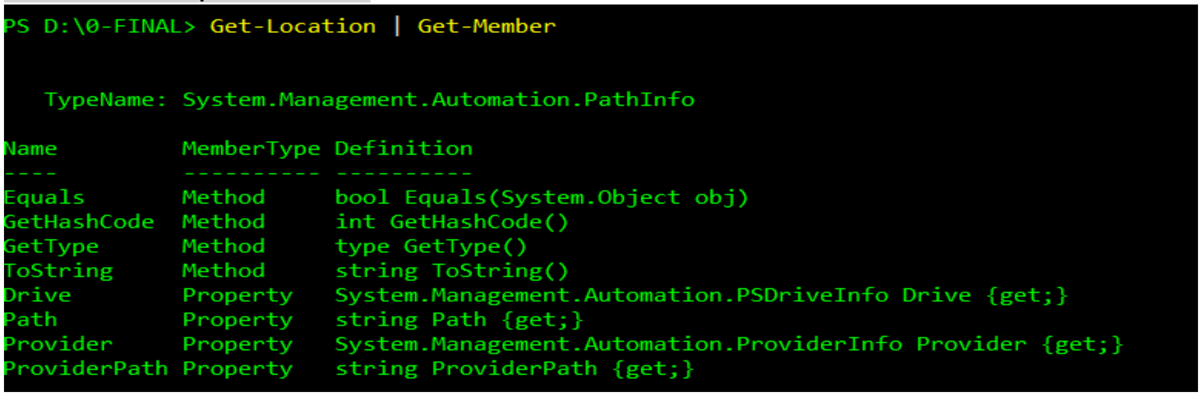
```
Get-Location
```



```
Windows PowerShell
PS D:\0-FINAL> Get-Location

Path
----
D:\0-FINAL
```

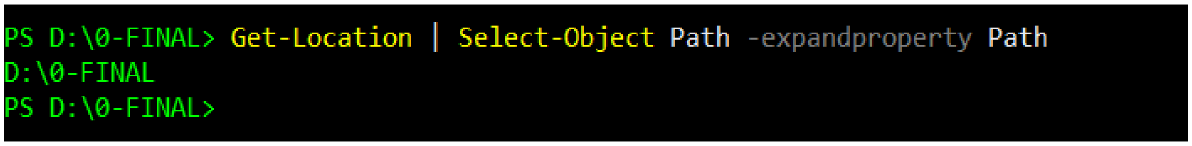
When running the above command, an object is returned. That object has it's own methods and properties. To view them, we will pipe the result to the **Get-Member** cmdlet:

```
Get-Location | Get-Member
```



```
PS D:\0-FINAL> Get-Location | Get-Member


   TypeName: System.Management.Automation.PathInfo

Name         MemberType Definition
----         ---------- ----------
Equals       Method     bool Equals(System.Object obj)
GetHashCode  Method     int GetHashCode()
GetType      Method     type GetType()
ToString     Method     string ToString()
Drive        Property   System.Management.Automation.PSDriveInfo Drive {get;}
Path         Property   string Path {get;}
Provider     Property   System.Management.Automation.ProviderInfo Provider {get;}
ProviderPath Property   string ProviderPath {get;}
```

We will use the **Get-Location**'s **Path** property value from the object being passed down the pipeline. The following will provide us with the **Path** property value and remove any empty lines from the command results:
**Get-Location | Select-Object Path -expandproperty Path**



```
PS D:\0-FINAL> Get-Location | Select-Object Path -expandproperty Path
D:\0-FINAL
PS D:\0-FINAL>
```

Before proceeding, ensure you create a directory in your present working directory named **LOGS**.
Now that we have the path to our scripts directory, we will improve our logfile naming convention by adding date and time to the logfile name. The result will be assigned to a variable.

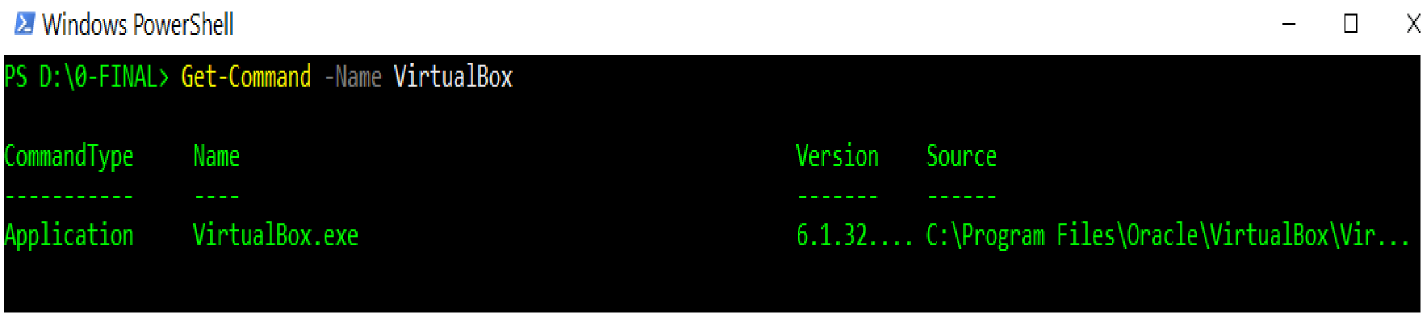
```
$logfile = (Get-Location | Select-Object Path -expandproperty Path) + "\LOGS\auto_start_vm_$(get-date -f yyyy-MM-dd-hhmmss).log"
```

We will start the logging process with the following:
**Start-Transcript -path "$logfile" -append**

Now, the logfile will be populated each time **Write-Host** is executed in the script.

Next, we will determine the path to the VirtualBox executable using the **Get-Command** cmdlet.

```
Get-Command -Name VirtualBox
```

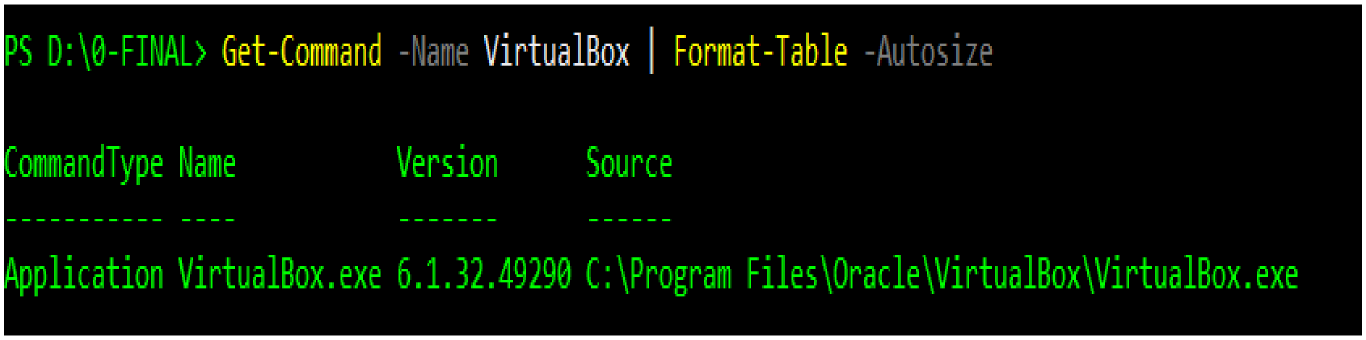


Windows PowerShell

```
PS D:\0-FINAL> Get-Command -Name VirtualBox

CommandType     Name                                               Version    Source
-----------     ----                                               -------    ------
Application     VirtualBox.exe                                     6.1.32.... C:\Program Files\Oracle\VirtualBox\Vir...
```

You will notice that the field value that we need (**Source**) is truncated. We will use the following to improve the readability of the tabular results.

```
Get-Command -Name VirtualBox | Format-Table -Autosize
```

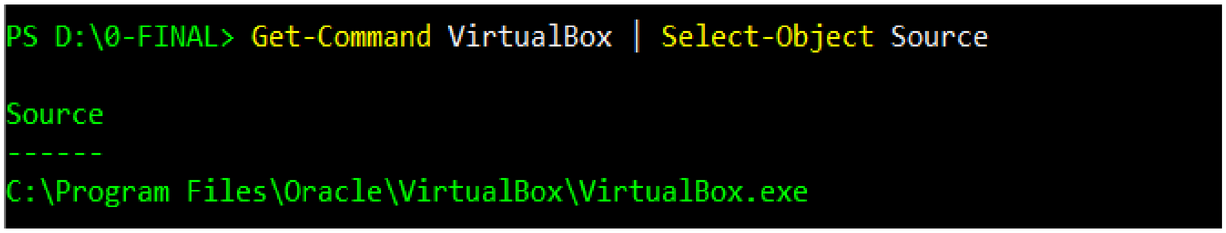


```
PS D:\0-FINAL> Get-Command -Name VirtualBox | Format-Table -Autosize

CommandType Name           Version       Source
----------- ----           -------       ------
Application VirtualBox.exe 6.1.32.49290 C:\Program Files\Oracle\VirtualBox\VirtualBox.exe
```

Using the **`Autosize`** parameter, when running **`Format-Table`**, column widths are calculated based on size of actual data.

The following will return an object with one property, **Source**.

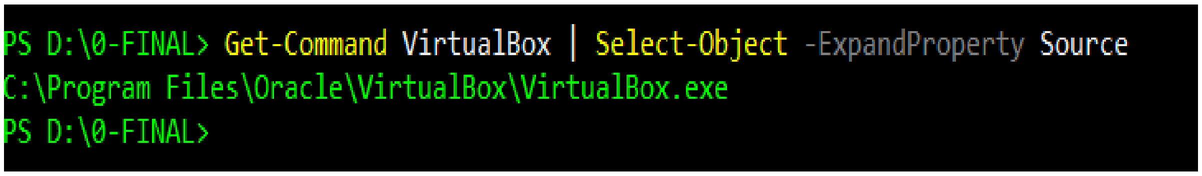
```
Get-Command VirtualBox | Select-Object Source
```



```
PS D:\0-FINAL> Get-Command VirtualBox | Select-Object Source

Source
------
C:\Program Files\Oracle\VirtualBox\VirtualBox.exe
```

Before we assign this value to a variable, we will need to extract the **Source** property value from the object being passed down the pipeline. The following will provide us with the **Source** property value and remove any empty lines from the command results:

```
Get-Command VirtualBox | Select-Object -ExpandProperty Source
```



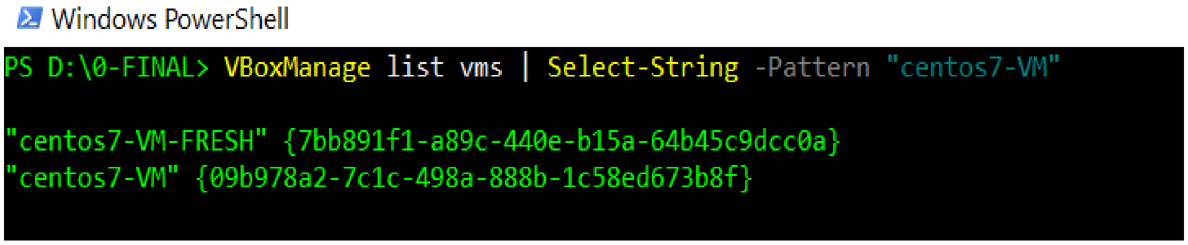
```
PS D:\0-FINAL> Get-Command VirtualBox | Select-Object -ExpandProperty Source
C:\Program Files\Oracle\VirtualBox\VirtualBox.exe
PS D:\0-FINAL>
```

Now we can assign the command result to a variable for later use.

```
$vbox = Get-Command VirtualBox | Select-Object -ExpandProperty Source
```

Before proceeding, I want to ensure that the VM name I specify exists. I will use the following:
*(**NOTE**: use the VM name you wish to start)*

```
VBoxManage list vms | Select-String -Pattern "centos7-VM"
```



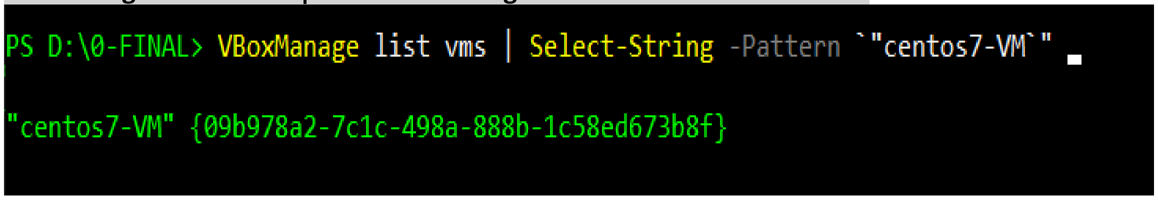
Windows PowerShell

```
PS D:\0-FINAL> VBoxManage list vms | Select-String -Pattern "centos7-VM"

"centos7-VM-FRESH" {7bb891f1-a89c-440e-b15a-64b45c9dcc0a}
"centos7-VM" {09b978a2-7c1c-498a-888b-1c58ed673b8f}
```

You will notice that I have two VMs whose names begin with "**centos7-VM**". PowerShell's **Select-String** cmdlet returned both of them, as demonstrated above.

I will use backticks to escape the double quotes to ensure that the double quotes are included in the search (**`"centos7-VM`"**).

```
VBoxManage list vms | Select-String -Pattern `"centos7-VM`"
```



```
PS D:\0-FINAL> VBoxManage list vms | Select-String -Pattern `"centos7-VM`"

"centos7-VM" {09b978a2-7c1c-498a-888b-1c58ed673b8f}
```

For each object returned by **Select-String**, we are finding the value of each match. Although this tutorial is not a deep-dive into PowerShell cmdlets, the default output of `Select-String` is a **MatchInfo** object, which includes detailed information (properties) about the matches. The `MatchInfo` object has a property called **Matches**, which contains a list of regular expression matches. For each of the matches in the list returned, we are using the **Value** property of the Match to get the actual value.

Now that we've confirmed that the VM we want to start exists, we will proceed.

Next, we will prompt the user for the virtual machine name to be started using the following:

```
$vmname = Read-Host "Enter a unique virtual machine name"
$vmname_mod = "`"" + $vmname + "`""
```

We will now ensure that the VM name entered is correct, as well as, check to see if it is already running.

```
$vm = VBoxManage list vms | Select-String -Pattern $vmname_mod | % { $_.Matches } | % { $_.Value }
$vm_up = VBoxManage list runningvms | Select-String -Pattern $vmname_mod | % { $_.Matches } | % { $_.Value }
| wc -l
```

Next, we will ensure that the VM has Guest Additions installed before proceeding by using **VBoxManage guestproperty**, to confirm that a valid value is returned by our query on the VM's Guest Additions Version.

```
# ensure Guest Additions installed before proceeding
if ((VBoxManage guestproperty get $vm /VirtualBox/GuestAdd/Version) -eq "No value set!") {
	Write-Host "VirtualBox's Guest Additions must be installed to auto-start $vmname."
	exit 1
} else {
	Write-Host "Confirmed that $vmname has Guest Additions installed."
	Write-Host "Proceeding ..."
}
```

Now we can use the **$vm_up** variable to determine if the VM is already running.

```
# ensure VM is not already running
if ($vm_up -eq 1) {
	Write-Host "$vmname is already running!"
	Write-Host "Goodbye."
	exit 1
# ensure user specified VM successfully located on system
} elseif ($vm) {
	Write-Host "$vmname exists. Proceeding..."
} else {
	Write-Host "VM $vmname does not exist."
	Write-Host "Goodbye."
	exit 1
}
```

After we confirm that the VM exists, and is not already running, I will be starting VirtualBox so that I have the management interface at my disposal. VirtualBox does **not** need to be started first. You can include the following in your script, or not, the choice is yours.

```
if (Get-Process VirtualBox -ErrorAction SilentlyContinue) {
	Write-Host "VirtualBox already running."
} else {
	start-process "$vbox"

	if (Get-Process VirtualBox -ErrorAction SilentlyContinue) {
		Write-Host "VirtualBox has successfully started."
	} else {
		Write-Host "Something went wrong during VirtualBox startup."
	}
}
```

You will notice that I have included **`-ErrorAction SilentlyContinue`** in our test. This is what's known as a **Common Parameter** of the Get-Process cmdlet. If VirtualBox did not start, instead of generating error messages in the PowerShell console, the command will continue execution and jump to the **`else`** clause of the **`if/else`** statement.

Now we are ready to start the virtual machine, whose name we assigned to the variable **`$vmname`**.

```
# NOTE: works without VirtualBox being started first
VBoxManage startvm $vmname
```

To verify that the previous command executed successfully, we perform another test.

```
# verify success of last command executed
if ( "$?" -eq "True") {
      Write-Host "$vm is booting up."
} else {
      Write-Host "Problem starting $vm"
}
```

This time we are using the PowerShell automatic variable **$?**. This variable contains the status of the last command executed: **True** if it succeeded and **False** if it failed.

The following block of code waits until the VM has completed the startup process. It also prints out boot duration in seconds. Using VBoxManage guestproperty, we are verifying that a value is returned by our query on the VM's **LoggedInUsers**. If **0** is returned, the VM is running, but no user has logged in. If "**No value set!**" is returned, the VM is not running.

```
# ensure virtual machine property set before proceeding (>= 0)
$n = 0
do {
      $n += 1
      Write-Progress -Activity "Launching VM $vmname" -SecondsRemaining $n
      Start-Sleep -s 1

} until ((VBoxManage guestproperty get $vm "/VirtualBox/GuestInfo/OS/LoggedInUsers") -ne "No value
set!")

Write-Host "$n seconds to boot $vmname"
```

Before launching an SSH connection to our running VM, we must ensure that the SSHD service is running on the VM.

For the code block below, we are using VBoxManage guestcontrol to confirm that the SSHD service is running. Make sure you substitute **your root user's password.**

```
Write-Host "Confirm that SSHD service is running on $vmname"

if ((VBoxManage --nologo guestcontrol $vm run --exe "/usr/bin/systemctl" --username 'root'
--password 'Pa$$w0rd' --wait-stdout -- systemctl/arg0 status sshd | grep "running" | wc -l) -eq 1)
{
      Write-Host "SSHD service is running on $vmname"
} else {
      Write-Host "SSHD service is NOT running on $vmname"
      exit 1
}
```

Now we can launch our SSH connection to our running VM. You will notice that I've created an XML file that stores the process id of the running PuTTY session. This will be used in my follow up tutorial, **Auto-Stop Virtual Machine Part 2**, where I create a PowerShell script to stop both the running VM and the opened PuTTY passwordless SSH session connected to the VM.

You will need to substitute:
a) where PuTTY is installed on your system
b) the name of your PuTTY Passwordless SSH connection

```
Write-Host "Establishing an SSH connection to $vmname."

# determine process id of newly started PuTTY SSH session
$putty_pid=(Start-Process -FilePath "C:\apps\putty.exe" -ArgumentList "-load centos7-VM" -
passthru).ID

if ($putty_pid) {
	Write-Host "You are now remotely connected to $vmname"

	# save PuTTY process id to file
	$putty_pid_file = (pwd | select Path -expandproperty Path) + "\LOGS\putty_pid_" + $vmname +
".xml"
	$putty_pid | Export-Clixml $putty_pid_file
} else {
	Write-Host "Problem establishing SSH connection to $vmname"
}
```

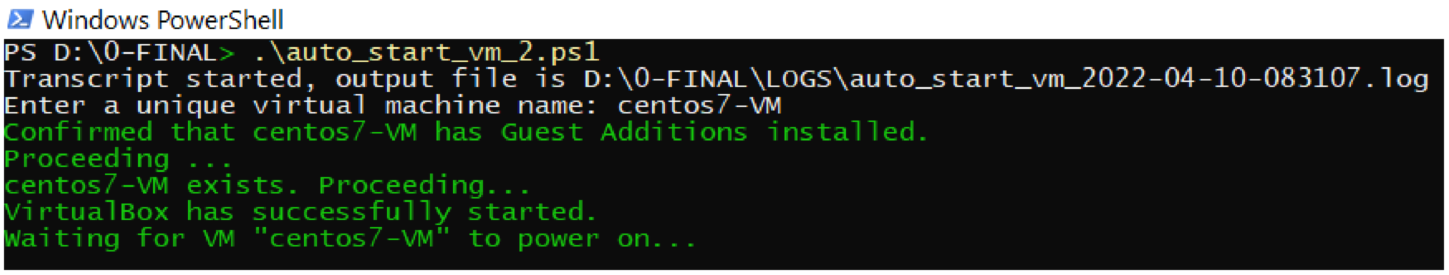
## Review Script

```
$logfile = (Get-Location | Select-Object Path -expandproperty Path) + "\LOGS\auto_start_vm_$(get-date -f
yyyy-MM-dd-hhmmss).log"

Start-Transcript -path "$logfile" -append

$vbox = Get-Command VirtualBox | Select Source -expandproperty Source

# prompt user for virtual machine name
$vmname = Read-Host "Enter a unique virtual machine name"
$vmname_mod = "`"" + $vmname + "`""

# locate VM specified by user
$vm = VBoxManage list vms | Select-String -Pattern $vmname_mod | % { $_.Matches } | % { $_.Value }
$vm_up = VBoxManage list runningvms | Select-String -Pattern $vmname_mod | % { $_.Matches } | % { $_.Value }
| wc -l

# ensure Guest Additions installed before proceeding
if ((VBoxManage guestproperty get $vm /VirtualBox/GuestAdd/Version) -eq "No value set!") {
	Write-Host "VirtualBox's Guest Additions must be installed to auto-start $vmname."
	exit 1
} else {
	Write-Host "Confirmed that $vmname has Guest Additions installed."
	Write-Host "Proceeding ..."
}

# ensure VM is not already running
if ($vm_up -eq 1) {
	Write-Host "$vmname is already running!"
	Write-Host "Goodbye."
	exit 1
# ensure user specified VM successfully located on system
} elseif ($vm) {
	Write-Host "$vmname exists. Proceeding..."
} else {
	Write-Host "VM $vmname does not exist."
	Write-Host "Goodbye."
	exit 1
}
```

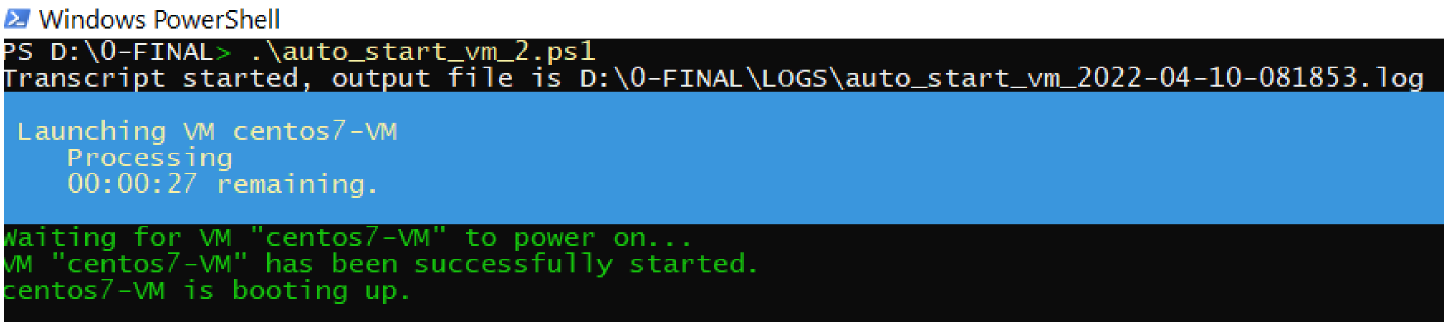
```
if (Get-Process VirtualBox -ErrorAction SilentlyContinue) {
       Write-Host "VirtualBox already running."
} else {
       start-process "$vbox"

       if (Get-Process VirtualBox -ErrorAction SilentlyContinue) {
              Write-Host "VirtualBox has successfully started."
       } else {
              Write-Host "Something went wrong during VirtualBox startup."
       }
}

# works without VirtualBox being started first
VBoxManage startvm $vmname

# verify success of last command executed
if ( "$?" -eq "True") {
       Write-Host "$vmname is booting up."
} else {
       Write-Host "Problem starting $vmname"
}

# ensure virtual machine property available before proceeding
$n = 0
do {
       $n += 1
       Write-Progress -Activity "Launching VM $vmname" -SecondsRemaining $n
       Start-Sleep -s 1

} until ((VBoxManage guestproperty get $vm "/VirtualBox/GuestInfo/OS/LoggedInUsers") -ne "No value set!")

Write-Host "$n seconds to boot $vmname"

Write-Host "Confirm that SSHD service is running on $vmname"

if ((VBoxManage --nologo guestcontrol $vm run --exe "/usr/bin/systemctl" --username 'root' --password
'Pa$$w0rd' --wait-stdout -- systemctl/arg0 status sshd | grep running | wc -l) -eq 1) {
       Write-Host "SSHD service is running on $vmname"
} else {
       Write-Host "SSHD service is NOT running on $vmname"
       exit 1
}

Write-Host "Establishing an SSH connection to $vmname."

# determine process id of newly started PuTTY SSH session
$putty_pid=(Start-Process -FilePath "C:\apps\putty.exe" -ArgumentList "-load CentOS7-VM" -passthru).ID

if ($putty_pid) {
       Write-Host "You are now remotely connected to $vmname"
       # save PuTTY process id to file
       $putty_pid_file = (pwd | select Path -expandproperty Path) + "\LOGS\putty_pid_" + $vmname + ".xml"
       $putty_pid | Export-Clixml $putty_pid_file
} else {
       Write-Host "Problem establishing SSH connection to $vmname"
}

Write-Host "Have a productive session!"
Stop-Transcript
```

## Execute Script

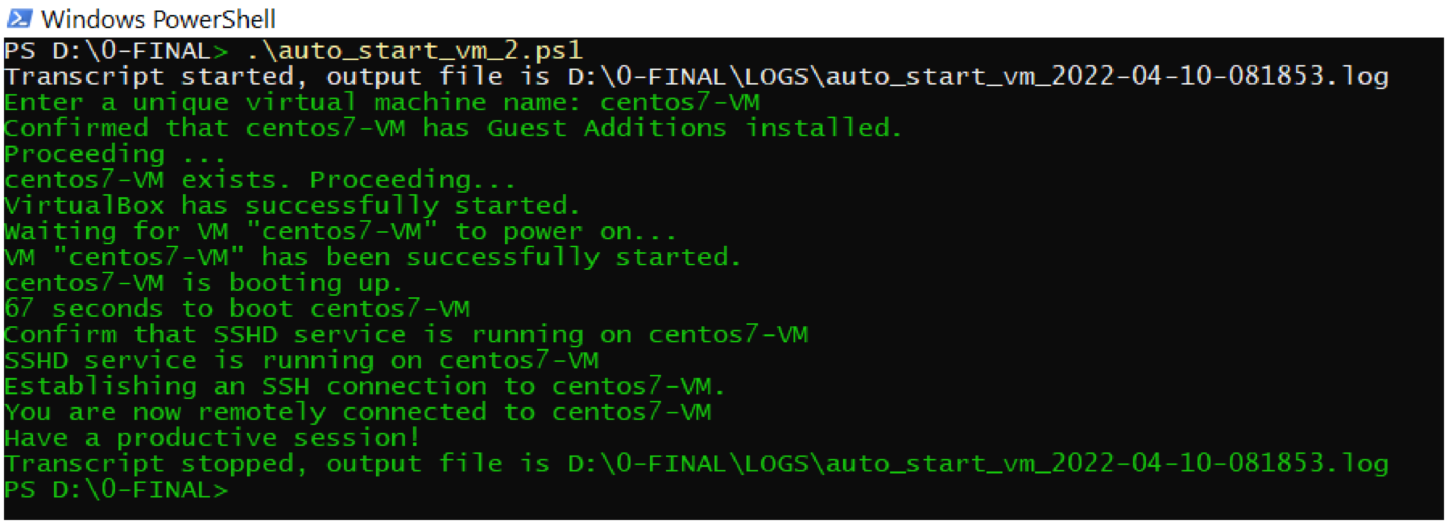
To execute the script, open a PowerShell console and change directory to the location of your script and execute the following:
**`.\auto_start_vm_2.ps1`**



```
PS D:\0-FINAL> .\auto_start_vm_2.ps1
Transcript started, output file is D:\0-FINAL\LOGS\auto_start_vm_2022-04-10-083107.log
Enter a unique virtual machine name: centos7-VM
Confirmed that centos7-VM has Guest Additions installed.
Proceeding ...
centos7-VM exists. Proceeding...
VirtualBox has successfully started.
Waiting for VM "centos7-VM" to power on...
```

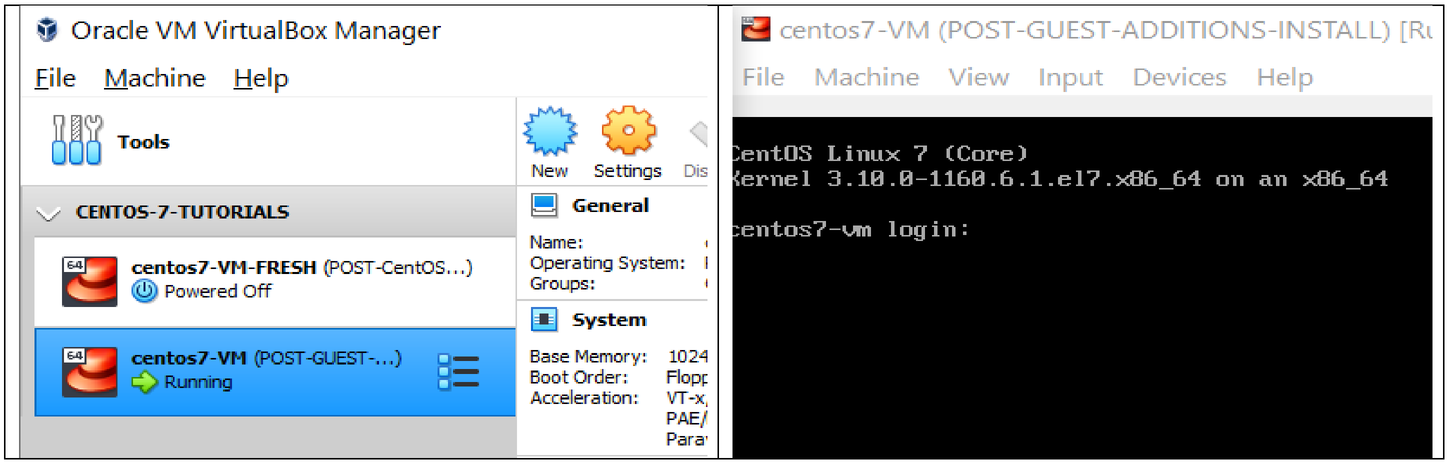


```
PS D:\0-FINAL> .\auto_start_vm_2.ps1
Transcript started, output file is D:\0-FINAL\LOGS\auto_start_vm_2022-04-10-081853.log

 Launching VM centos7-VM
    Processing
    00:00:27 remaining.

Waiting for VM "centos7-VM" to power on...
VM "centos7-VM" has been successfully started.
centos7-VM is booting up.
```



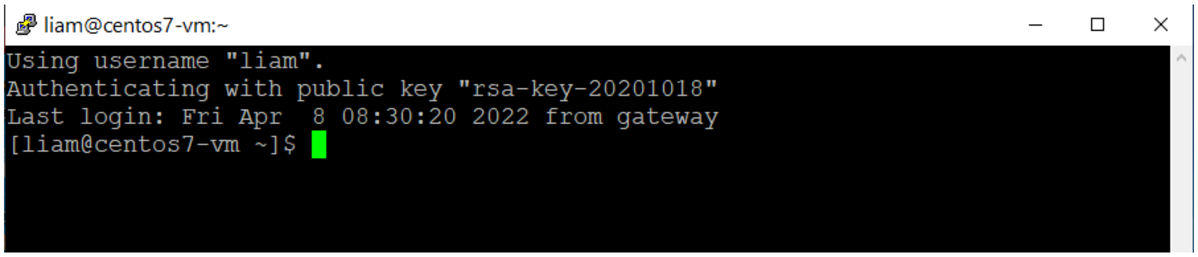
```
PS D:\0-FINAL> .\auto_start_vm_2.ps1
Transcript started, output file is D:\0-FINAL\LOGS\auto_start_vm_2022-04-10-081853.log
Enter a unique virtual machine name: centos7-VM
Confirmed that centos7-VM has Guest Additions installed.
Proceeding ...
centos7-VM exists. Proceeding...
VirtualBox has successfully started.
Waiting for VM "centos7-VM" to power on...
VM "centos7-VM" has been successfully started.
centos7-VM is booting up.
67 seconds to boot centos7-VM
Confirm that SSHD service is running on centos7-VM
SSHD service is running on centos7-VM
Establishing an SSH connection to centos7-VM.
You are now remotely connected to centos7-VM
Have a productive session!
Transcript stopped, output file is D:\0-FINAL\LOGS\auto_start_vm_2022-04-10-081853.log
PS D:\0-FINAL>
```

VirtualBox successfully started and so did the virtual machine.


Oracle VM VirtualBox Manager
File Machine Help
Tools
CENTOS-7-TUTORIALS
centos7-VM-FRESH (POST-CentOS...)
Powered Off
centos7-VM (POST-GUEST-...)
Running
New
Settings
Dis
General
Name:
Operating System:
Groups:
System
Base Memory: 1024
Boot Order: Flopp
Acceleration: VT-x, PAE/ Para
centos7-VM (POST-GUEST-ADDITIONS-INSTALL) [Ru
File Machine View Input Devices Help

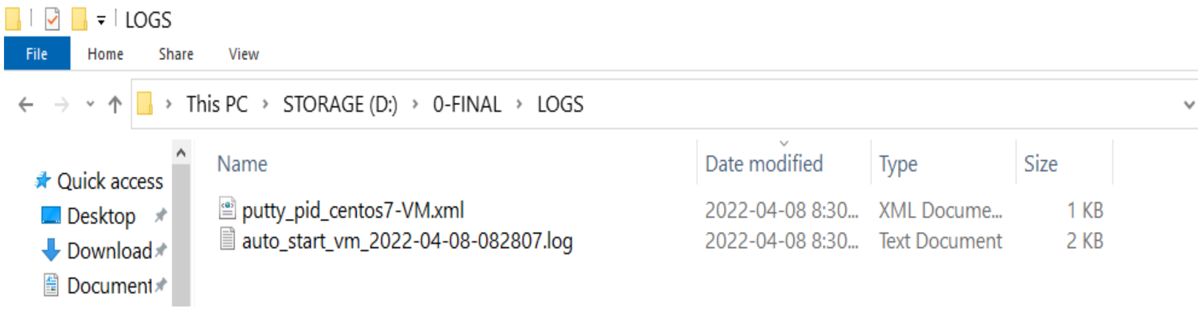
```
CentOS Linux 7 (Core)
Kernel 3.10.0-1160.6.1.el7.x86_64 on an x86_64

centos7-vm login:
```

The PuTTY passwordless SSH connection was also established.

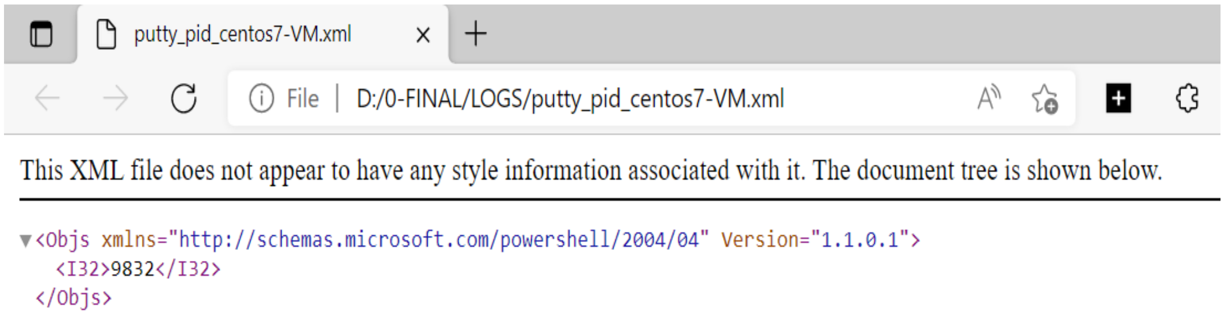


```
Using username "liam".
Authenticating with public key "rsa-key-20201018"
Last login: Fri Apr  8 08:30:20 2022 from gateway
[liam@centos7-vm ~]$
```

Finally, the XML file storing the PuTTY session's process id was created, as well as, the logfile.

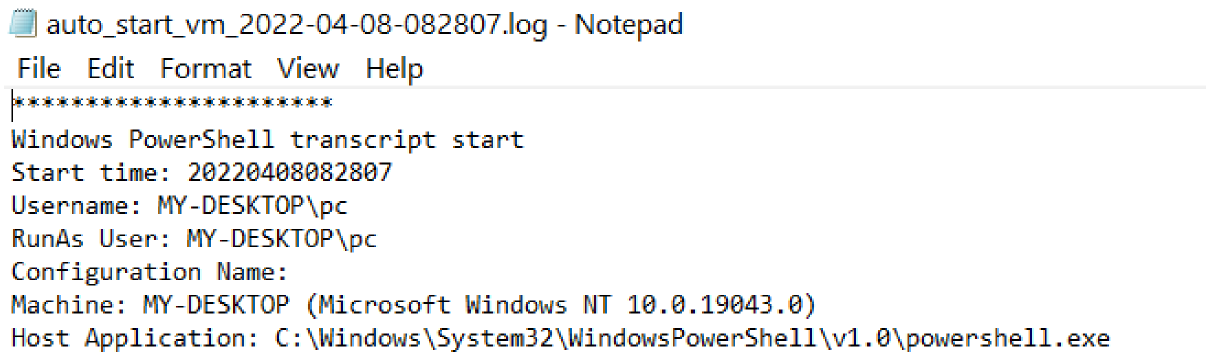


| Name | Date modified | Type | Size |
|---|---|---|---|
| putty_pid_centos7-VM.xml | 2022-04-08 8:30... | XML Docume... | 1 KB |
| auto_start_vm_2022-04-08-082807.log | 2022-04-08 8:30... | Text Document | 2 KB |



This XML file does not appear to have any style information associated with it. The document tree is shown below.

```
▼<Objs xmlns="http://schemas.microsoft.com/powershell/2004/04" Version="1.1.0.1">
   <I32>9832</I32>
 </Objs>
```


auto_start_vm_2022-04-08-082807.log - Notepad
File Edit Format View Help

```
**********************
Windows PowerShell transcript start
Start time: 20220408082807
Username: MY-DESKTOP\pc
RunAs User: MY-DESKTOP\pc
Configuration Name:
Machine: MY-DESKTOP (Microsoft Windows NT 10.0.19043.0)
Host Application: C:\Windows\System32\WindowsPowerShell\v1.0\powershell.exe
```

Hopefully, you've enjoyed completing this tutorial and found it helpful.

Now that you can auto-start your VM along with a PuTTY passwordless SSH connection, you might want to see my other tutorial, where I create a PowerShell script to stop both the running VM and the opened PuTTY passwordless SSH session, **Auto-Stop Virtual Machine Part 2**, accessible **here**.

If you would like to see my other tutorials, they can be accessed **here**.

Back to top