

Auto-Stop Virtual Machine Part 2

This is a continuation of my previous tutorial **Auto-Start Virtual Machine Part 2**, accessible [here](#). In that tutorial we created a PowerShell script to automatically start an existing VirtualBox virtual machine with Guest Additions installed, ensure the SSHD service was running on the guest VM before launching a PuTTY passwordless SSH connection.

In this tutorial I will be creating a PowerShell script to automatically stop a running VirtualBox virtual machine that has Guest Additions installed. After detecting the VM's Guest Additions version, we will check whether an active PuTTY SSH connection exists and, if so, close it. Then, using [VBoxManage guestcontrol](#) we will shutdown the running 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 this tutorial [GitBash](#) installation is not required. However, I like having access to Unix/Linux utilities, in a Windows environment, that do not have the corresponding aliased PowerShell cmdlets. [GitBash](#) also provide the core [git](#) commands needed for [GitHub](#) repository management.

Refer to the prerequisites listed below to complete this tutorial.

Prerequisites

- Running VirtualBox VM with Guest Additions installed
- Running PuTTY client for managing SSH sessions
- SSH keypair
- GitBash installed (optional)

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_stop_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.

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 stop exists, we will proceed.

Next, we will prompt the user for the virtual machine name to be stopped 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 }
| Measure-Object -line | select-object -expandproperty Lines
```

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 running. Next, we are checking for the existence of a PuTTY process id file, closing the open SSH connection (if it exists) and deleting the process id file. Finally, we shutdown the VM.

```
# check if VM is running
if ($vm_up -eq 1) {
    Write-Host "Detecting PuTTY SSH connection."
    $putty_pid_file = (pwd | select Path -expandproperty Path) + "\LOGS\putty_pid_" + $vmname + ".xml"
    # check for existence PuTTY process id file
    if ( Test-Path $putty_pid_file -PathType Leaf ) {
        $putty_pid = Import-Clixml $putty_pid_file
        # if running, stop PuTTY session
        if (Get-Process -Id $putty_pid -ErrorAction SilentlyContinue) {
            Write-Host "Closing PuTTY SSH connection."
            Stop-Process -Id $putty_pid
        }
        # delete PuTTY process id file
        Remove-Item $putty_pid_file
    } else {
        Write-Host "PuTTY pid file does not exist"
        Write-Host "No active PuTTY SSH connection."
    }
}
```

```

# ensure virtual machine property available before proceeding
$n = 0
do {
    $n += 1
} until ((VBoxManage guestproperty get $vm "/VirtualBox/GuestInfo/OS/LoggedInUsers") -ne
"No value set!")

#Write-Host "Using VBoxManage guestcontrol to shutdown $vmname"
VBoxManage --nologo guestcontrol $vm run --exe "/usr/sbin/shutdown" --username 'root' --
password 'Pa$$w0rd' --wait-stdout -- -h now

} else {
    Write-Host "VM $vmname is not running."
    Write-Host "Goodbye."
    exit 1
}

```

We now want to wait until the VM has completely shutdown before proceeding.

Write-Host "Stopping VM \$vmname"

```

# ensure user specified VM has completely shutdown
$n = 0
do {
    $n += 1
} until ((VBoxManage list runningvms | Select-String -Pattern $vm | Select-Object -ExpandProperty
Line | Measure-Object -line | Select-Object -ExpandProperty Lines) -eq 0)

```

After the VM has completely shutdown, we check whether any other VMs are running before closing the VirtualBox Management Interface.

```

# check for other running VMs
$vms = (VBoxManage list runningvms | Measure-Object -line | Select-Object -ExpandProperty Lines)

# if no other VMs are running, shutdown VirtualBox
if (!(Get-Process VirtualBox -ErrorAction SilentlyContinue)) {
    Write-Host "VirtualBox NOT running."
} else {
    if ($vms -gt 0) {
        Write-Host "There is still at least one running VM."
        Write-Host "Will not be closing VirtualBox."
    } else {
        Write-Host "Closing VirtualBox"
        Stop-Process -ProcessName VirtualBox
    }
}
}

```

We are now ready to review the contents of the entire script.

Review Script

```
$logfile = (pwd | select Path -expandproperty Path) + "\LOGS\auto_stop_vm_$(get-date -f yyyy-MM-dd-
hhmmss).log"
```

```
Start-Transcript -path "$logfile" -append
```

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

```
$vm = VBoxManage list vms | Select-String -Pattern $vmname_mod | % { $_.Matches } | % { $_.Value }
$vm_up = VBoxManage list runningvms | Select-String -Pattern $vmname_mod | % { $_.Matches } | % { $_.Value }
| Measure-Object -line | select-object -expandproperty Lines

```

```

# 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-stop $vmname."
    exit 1
} else {
    Write-Host "Confirmed that $vmname has Guest Additions installed."
    Write-Host "Proceeding ..."
}

if ($vm_up -eq 1) {
    Write-Host "Detecting PuTTY SSH connection."
    $putty_pid_file = (pwd | select Path -expandproperty Path) + "\LOGS\putty_pid_" + $vmname + ".xml"

    if ( Test-Path $putty_pid_file -PathType Leaf ) {
        $putty_pid = Import-Clixml $putty_pid_file
        # if running, stop PuTTY session
        if (Get-Process -Id $putty_pid -ErrorAction SilentlyContinue) {
            Write-Host "Closing PuTTY SSH connection."
            Stop-Process -Id $putty_pid
        }
        # delete PuTTY process id file
        Remove-Item $putty_pid_file
    } else {
        Write-Host "PuTTY pid file does not exist"
        Write-Host "No active PuTTY SSH connection."
    }

    # ensure virtual machine property available before proceeding
    $n = 0
    do {
        $n += 1
    } until ((VBoxManage guestproperty get $vm "/VirtualBox/GuestInfo/OS/LoggedInUsers") -ne "No value
set!")

    VBoxManage --nologo guestcontrol $vm run --exe "/usr/sbin/shutdown" --username 'root' --password
'Pa$$w0rd' --wait-stdout -- -h now

} else {
    Write-Host "VM $vmname is not running."
    Write-Host "Goodbye."
    exit 1
}

Write-Host "Stopping VM $vmname"

# ensure user specified VM has completely shutdown
$n = 0
do {
    $n += 1
} until ((VBoxManage list runningvms | Select-String -Pattern $vm | Select-Object -ExpandProperty Line |
Measure-Object -line | Select-Object -ExpandProperty Lines) -eq 0)

# check for other running VMs
$vms = (VBoxManage list runningvms | Measure-Object -line | Select-Object -ExpandProperty Lines)

if (!(Get-Process VirtualBox -ErrorAction SilentlyContinue)) {
    Write-Host "VirtualBox NOT running."
} else {
    if ($vms -gt 0) {
        Write-Host "There is still at least one running VM."
        Write-Host "Will not be closing VirtualBox."
    } else {
        Write-Host "Closing VirtualBox"
        Stop-Process -ProcessName VirtualBox
    }
}

Stop-Transcript

```

Execute Script

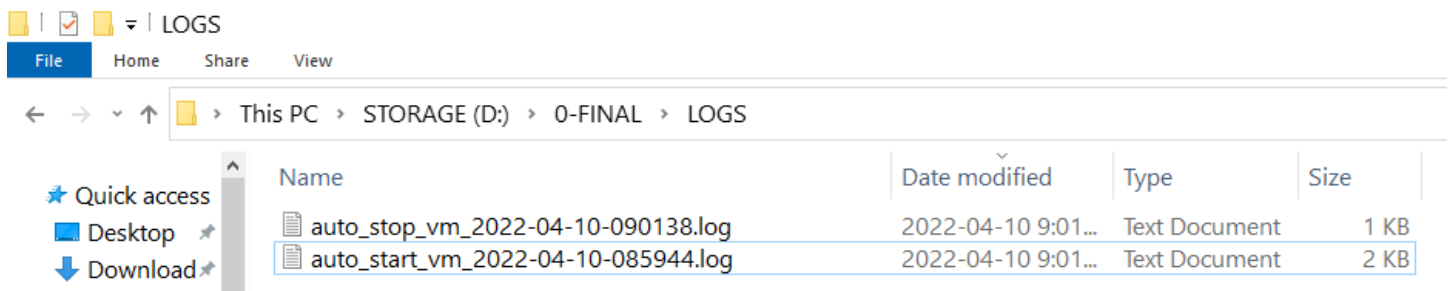
and change directory to the location of your script and execute the following:

```
.\auto_stop_vm_2.ps1
```

```
Windows PowerShell
PS D:\0-FINAL> .\auto_stop_vm_2.ps1
Transcript started, output file is D:\0-FINAL\LOGS\auto_stop_vm_2022-04-10-083336.log
Enter a unique virtual machine name: centos7-VM
Confirmed that centos7-VM has Guest Additions installed.
Proceeding ...
Detecting PuTTY SSH connection.
Closing PuTTY SSH connection.
Stopping VM centos7-VM
Closing VirtualBox
Transcript stopped, output file is D:\0-FINAL\LOGS\auto_stop_vm_2022-04-10-083336.log
PS D:\0-FINAL>
```

First, the PuTTY SSH connection was closed and it's process id file was deleted. Then, the VM was shutdown and, finally, VirtualBox was closed.

We can confirm that the PuTTY process id file was deleted and that the new stop logfile were created by opening our **LOGS** directory. Below, you will also notice my existing start logfile from my previous run of **auto_start_vm_2.ps1**.



```
auto_stop_vm_2022-04-10-090138.log - Notepad
File Edit Format View Help
*****
Windows PowerShell transcript start
Start time: 20220410090138
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
Process ID: 7048
```

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

After completing both my advanced automation tutorials to start (accessible [here](#)) and stop a VM, along with the starting, or stopping, of a PuTTY passwordless SSH connection, you might want to see some of my other tutorials where I demonstrate the installation, and administration of, Linux using a Cloud instance. My Linux tutorials can be accessed [here](#), while my Cloud tutorials can be accessed [here](#).

If you would like to see my other tutorials, they can be accessed [here](#).

[Back to top](#)