What's in Your PowerShell Profile?

“What’s in Your Profile?”

This is a question you see often in PowerShell blogs and forums. PowerShell profiles can be a powerful way to automate your command line environment and most importantly save typing and time.

In my previous post I went over how to make a portable PowerShell profile. In this post I’ll go over what I have in my profile scripts and why.

The Main Profile

In this profile script I configure most of the settings that will apply to both the PowerShell console and the ISE:

```
### Import Credentials
$SAMMattAdmin = (Import-Clixml "$DirScripts\Creds\cred_SAM_mattadmin_$Env:ComputerName.xml")
$UPNattAdmin = (Import-Clixml "$DirScripts\Creds\cred_UPN_mattadmin_$Env:ComputerName.xml")

### Extend the module path to include Onedrive folder
$Env:PSModulePath = $Env:PSModulePath -replace 'c:\\Users\\matt\\My Documents\\WindowsPowerShell\\Modules',''
$Env:PSModulePath += ";$DirScripts\Modules"

### Custom prompt
. "$PSScriptRoot\prompt.ps1"

### Import commonly used modules
Import-Module ActiveDirectory
Import-Module ADLibrary
Import-Module MSOLLibrary
Import-Module ServerAdminLibrary
Import-Module GeneralLibrary

```

```
switch ($Env:PROCESSOR_ARCHITECTURE)
{
'x86' {$AzureModulePath = "$env:ProgramFiles\Microsoft SDKs\Azure\powershell\ServiceManagement\Azure\Azure.psd1"}
'AMD64' {$AzureModulePath = "${env:ProgramFiles(x86)}\Microsoft SDKs\Azure\powershell\ServiceManagement\Azure\Azure.psd1"}
}

### Create a Cim session to PS01v for printer operations
$CimPs01 = New-CimSession -ComputerName ps01 -Credential $SAMMattAdmin

### set default values for commonly used parameters
. "$PSScriptRoot\PSDefaultParameterValues.ps1"

### Customize the ISE
if ($psise)
{
Import-Module ISELibrary
. "$PSScriptRoot\ISEConfig.ps1"
}
```

view raw 1.ps1 hosted with ❤ by GitHub

Let’s walk through this bit-by-bit. The first two lines import credential objects that I have previously saved using the New-SavedCredential function and saves them in variables. This saves the username and password securely in an XML file that represents a PowerShell credential object. I typically run powershell with a standard user account and use these credentials in commands that require more privilege. I save the credentials in SAM and UPN formats - SAM format is for things like domain servers and Active Directory, and the UPN credentials can be used against web services like Office 365.

```
### Import Credentials
$SAMMattAdmin = (Import-Clixml "$DirScripts\Creds\cred_SAM_mattadmin_$Env:ComputerName.xml")
$UPNattAdmin = (Import-Clixml "$DirScripts\Creds\cred_UPN_mattadmin_$Env:ComputerName.xml")
```

view raw 2.ps1 hosted with ❤ by GitHub

Next I extend the PowerShell module path to include my script library; this adds quick access to modules stored in OneDrive. I also remove my local documents modules folder from the modulepath variable. I do this because I often work over a VPN connection and my documents are stored on a file server in our datacenter. This can mean that automatic module enumeration is very slow over the WAN connection. This cripples tab-completion and Intellisense because it constantly checks your module path to discover commands and parameters.

```
### Extend the module path to include Onedrive folder
$Env:PSModulePath = $Env:PSModulePath -replace 'c:\\Users\\matt\\My Documents\\WindowsPowerShell\\Modules',''
$Env:PSModulePath += ";$DirScripts\Modules"
```

view raw 3.ps1 hosted with ❤ by GitHub

Next in line is a prompt function. When you create a function named prompt, the text output of that function will be used as the command-line prompt. I like mine with some color and it also displays the execution time of the last command run.

```
### Custom prompt
. "$PSScriptRoot\prompt.ps1"
```

view raw 4.ps1 hosted with ❤ by GitHub

```
function Get-LastExecutionTime
{
$LastCmd = Get-History -Count 1

$ExecTime = $LastCmd.EndExecutionTime - $LastCmd.StartExecutionTime
'{0:00}:{1:00}:{2:00}' -f $ExecTime.Minutes, $ExecTime.Seconds, $ExecTime.Milliseconds
}

function prompt {
if (((get-location).path).length -gt 40)
{
Write-Host "$(Get-Location)" -NoNewline -ForegroundColor Yellow
Write-Host "[$(Get-LastExecutionTime)]" -ForegroundColor Green
Write-Host $('&gt;' * ($NestedPromptLevel + 1)) -NoNewline -ForegroundColor Green
return ' '
}
else
{
Write-Host "$(Get-Location)" -NoNewline -ForegroundColor yellow
Write-Host "[$(Get-LastExecutionTime)]" -NoNewline -ForegroundColor Green
Write-Host $('&gt;' * ($NestedPromptLevel + 1)) -NoNewline -ForegroundColor Green
return ' '
}
}
```

view raw 5.ps1 hosted with ❤ by GitHub

Next I load several modules that I work with on a regular basis. Since PowerShell version 3.0 this is not really necessary since modules when auto-load when they are called upon, but I prefer that they load at startup to save time later. Besides that I still sometimes run PowerShell version 2.0 which does not auto-load modules, so this ensures the right modules are loaded when I run PowerShell. I have recently begun using the Windows Azure module but I don't really want to load it when I launch the shell, so I save the path to the module in a variable and call that later when I need to access the Azure cmdlets.

```
### Import commonly used modules
Import-Module ActiveDirectory
Import-Module ADLibrary
Import-Module MSOLLibrary
Import-Module ServerAdminLibrary
Import-Module GeneralLibrary

switch ($Env:PROCESSOR_ARCHITECTURE)
{
'x86' {$AzureModulePath = "$env:ProgramFiles\Microsoft SDKs\Azure\powershell\ServiceManagement\Azure\Azure.psd1"}
'AMD64' {$AzureModulePath = "${env:ProgramFiles(x86)}\Microsoft SDKs\Azure\powershell\ServiceManagement\Azure\Azure.psd1"}
}
```

view raw 6.ps1 hosted with ❤ by GitHub
I manage a print server on a daily basis and use cmdlets in the PrintManagement module to do so. These modules leverage WMI/CIM to retrieve and configure printer settings. For quick access I like to have a CIM session to my print server already open when I load PowerShell. I use one of my imported admin credentials to establish this connection.

```
### Create a Cim session to PS01v for printer operations
$CimPs01 = New-CimSession -ComputerName ps01 -Credential $SAMMattAdmin
```

view raw 7.ps1 hosted with ❤ by GitHub
I use the $PSDefaultParameterValues automatic variable to configure default values when I run certain commands. This is a huge time-saver if you find yourself typing the same parameter values in on a regular basis. An example of this is setting the -CimSession parameter of all of the PrintManagement cmdlets with a value of $CimPs01 from the previous line in my profile script. This makes sure that I am always running printer cmdlets against the right computer without having to enter the parameter at the command line.

```
### set default values for commonly used parameters
. "$PSScriptRoot\PSDefaultParameterValues.ps1"
{% endhighlight %}

{% highlight Powershell %}
$PSDefaultParameterValues = @{
'Get-Printer:CimSession' = $Cimps01
'Get-PrinterDriver:CimSession' = $Cimps01
'Get-PrinterPort:CimSession:CimSession' = $Cimps01

'Export-Csv:NoTypeInformation' = $true
}
```

view raw 8.ps1 hosted with ❤ by GitHub
Lastly, if the current PowerShell host is the ISE editor, I import a module called ISELibrary with a collection of functions for interacting with the ISE. I then run another profile script that makes some changes like the default pane view and loading custom add-ins.

```
### Customize the ISE
if ($psise)
{
Import-Module ISELibrary
. "$PSScriptRoot\ISEConfig.ps1"
}
```

view raw 9.ps1 hosted with ❤ by GitHub

```
$psISE.options.SelectedScriptPaneState = 'Maximized'
$psISE.Options.ShowDefaultSnippets = $false
Import-IseSnippet -Path "$Dirscripts\Snippets\" -Recurse

### Load Add-ons
. "$PSScriptRoot\load-addons.ps1"

Remove-ISEUntitled
Import-ISEEditorState -Path "$Dirscripts\temp"
```

view raw 10.ps1 hosted with ❤ by GitHub

Other scripts

The ISEConfig script above also calls another profile script - load-addons.ps1. This script uses some of the functions in my ISELibrary module to create ISE add-ons. These add-ons take advantage of the extensibility of the ISE object model and allow you to create ISE menu items that run PowerShell code and can be called with a keyboard shortcut.

```
$psTab = $psISE.CurrentPowerShellTab.addonsmenu
$IseLibrary = $psTab.Submenus.Add('IseLibrary',$null,$null)
$null = $IseLibrary.Submenus.Add('Toggle Font',{Switch-ISEFont},'Ctrl+Alt+F')
$null = $IseLibrary.Submenus.Add('Toggle Font',{Switch-ISEFontSize},'Ctrl+Alt+S')
$null = $IseLibrary.Submenus.Add('Remove Trailing Blanks', {Remove-TrailingBlanks}, 'Ctrl+Alt+T')
$null = $IseLibrary.Submenus.Add('Remove Blank Lines', {Remove-BlankLines}, 'Ctrl+Alt+B')
```

```
$null = $IseLibrary.Submenus.Add('Save Editor State and Exit',{Save-ISEEditorStateAndExit},'Ctrl+Alt+E')
Remove-Variable -Name psTab, IseLibrary
```

view raw 11.ps1 hosted with ❤ by GitHub

Well, that's my PowerShell profile! I hope this wasn't too long-winded but instead illustrates how flexible and dynamic PowerShell profiles can be.

So, what's in your profile?