# about_Profiles - PowerShell

By sdwheeler

Archived: 2026-04-05 18:08:26 UTC

## Short description

Describes how to create and use a PowerShell profile.

## Long description

You can create a PowerShell profile to customize your environment and add session-specific elements to every PowerShell session that you start.

A PowerShell profile is a script that runs when PowerShell starts. You can use the profile as a startup script to customize your environment. You can add commands, aliases, functions, variables, modules, PowerShell drives and more. You can also add other session-specific elements to your profile so they're available in every session without having to import or re-create them.

PowerShell supports several profiles for users and host programs. However, it doesn't create the profiles for you.

## Profile types and locations

PowerShell supports several profile files that are scoped to users and PowerShell hosts. You can have any or all these profiles on your computer.

The PowerShell console supports the following basic profile files. These file paths are the default locations.

- All Users, All Hosts
  - Windows - `$PSHOME\Profile.ps1`
  - Linux - `/opt/microsoft/powershell/7/profile.ps1`
  - macOS - `/usr/local/microsoft/powershell/7/profile.ps1`
- All Users, Current Host
  - Windows - `$PSHOME\Microsoft.PowerShell_profile.ps1`
  - Linux - `/opt/microsoft/powershell/7/Microsoft.PowerShell_profile.ps1`
  - macOS - `/usr/local/microsoft/powershell/7/Microsoft.PowerShell_profile.ps1`
- Current User, All Hosts
  - Windows - `$HOME\Documents\PowerShell\Profile.ps1`
  - Linux - `~/.config/powershell/profile.ps1`
  - macOS - `~/.config/powershell/profile.ps1`
- Current user, Current Host
  - Windows - `$HOME\Documents\PowerShell\Microsoft.PowerShell_profile.ps1`

    - Linux - `~/.config/powershell/Microsoft.PowerShell_profile.ps1`
    - macOS - `~/.config/powershell/Microsoft.PowerShell_profile.ps1`

The profile scripts are executed in the order listed. This means that changes made in the **AllUsersAllHosts** profile can be overridden by any of the other profile scripts. The **CurrentUserCurrentHost** profile always runs last. In PowerShell Help, the **CurrentUserCurrentHost** profile is the profile most often referred to as *your PowerShell profile*.

Other programs that host PowerShell can support their own profiles. For example, Visual Studio Code (VS Code) supports the following host-specific profiles.

- All users, Current Host - `$PSHOME\Microsoft.VSCode_profile.ps1`
- Current user, Current Host - `$HOME\Documents\PowerShell\Microsoft.VSCode_profile.ps1`

The profile paths include the following variables:

- The `$PSHOME` variable stores the installation directory for PowerShell
- The `$HOME` variable stores the current user's home directory

Note

In Windows, the location of the `Documents` folder can be changed by folder redirection or OneDrive. We don't recommend redirecting the `Documents` folder to a network share or including it in OneDrive. Redirecting the folder can cause modules to fail to load and create errors in your profile scripts. For information about removing the `Documents` folder from OneDrive management, consult the OneDrive documentation.

## The $PROFILE variable

The `$PROFILE` automatic variable stores the paths to the PowerShell profiles that are available in the current session.

To view a profile path, display the value of the `$PROFILE` variable. You can also use the `$PROFILE` variable in a command to represent a path.

The `$PROFILE` variable stores the path to the "Current User, Current Host" profile. The other profiles are saved in note properties of the `$PROFILE` variable.

For example, the `$PROFILE` variable has the following values in the Windows PowerShell console.

- Current User, Current Host - `$PROFILE`
- Current User, Current Host - `$PROFILE.CurrentUserCurrentHost`
- Current User, All Hosts - `$PROFILE.CurrentUserAllHosts`
- All Users, Current Host - `$PROFILE.AllUsersCurrentHost`
- All Users, All Hosts - `$PROFILE.AllUsersAllHosts`

Because the values of the `$PROFILE` variable change for each user and in each host application, ensure that you display the values of the profile variables in each PowerShell host application that you use.

To see the current values of the `$PROFILE` variable, type:

```
$PROFILE | Select-Object *
```

You can use the `$PROFILE` variable in many commands. For example, the following command opens the "Current User, Current Host" profile in Notepad:

```
notepad $PROFILE
```

The following command determines whether an "All Users, All Hosts" profile has been created on the local computer:

```
Test-Path -Path $PROFILE.AllUsersAllHosts
```

## How to create a profile

To create a PowerShell profile, use the following command format:

```
if (!(Test-Path -Path <profile-name>)) {
 New-Item -ItemType File -Path <profile-name> -Force
}
```

For example, to create a profile for the current user in the current PowerShell host application, use the following command:

```
if (!(Test-Path -Path $PROFILE)) {
 New-Item -ItemType File -Path $PROFILE -Force
}
```

In this command, the `if` statement prevents you from overwriting an existing profile. Replace the value of the `$PROFILE` variable with the path to the profile file that you want to create.

Note

To create "All Users" profiles in Windows Vista and later versions of Windows, start PowerShell with the **Run as administrator** option.

## How to edit a profile

You can open any PowerShell profile in a text editor, such as Notepad.

To open the profile of the current user in the current PowerShell host application in Notepad, type:

```
notepad $PROFILE
```

To open other profiles, specify the profile name. For example, to open the profile for all the users of all the host applications, type:

```
notepad $PROFILE.AllUsersAllHosts
```

To apply the changes, save the profile file, and then restart PowerShell.

## How to choose a profile

If you use multiple host applications, put the items that you use in all the host applications into your `$PROFILE.CurrentUserAllHosts` profile. Put items that are specific to a host application, such as a command that sets the background color for a host application, in a profile that's specific to that host application.

If you are an administrator who is customizing PowerShell for many users, follow these guidelines:

- Store the common items in the `$PROFILE.AllUsersAllHosts` profile
- Store items that are specific to a host application in `$PROFILE.AllUsersCurrentHost` profiles that are specific to the host application
- Store items for particular users in the user-specific profiles

Be sure to check the host application documentation for any special implementation of PowerShell profiles.

## How to use a profile

Many of the items that you create in PowerShell and most commands that you run affect only the current session. When you end the session, the items are deleted.

The session-specific commands and items include PowerShell variables, environment variables, aliases, functions, commands, and PowerShell modules that you add to the session.

To save these items and make them available in all future sessions, add them to a PowerShell profile.

Another common use for profiles is to save frequently used functions, aliases, and variables. When you save the items in a profile, you can use them in any applicable session without recreating them.

## How to start a profile

When you open the profile file, it's blank. However, you can fill it with the variables, aliases, and commands that you use frequently.

Here are a few suggestions to get you started.

### Add a function that lists aliases for any cmdlet

```
 function Get-CmdletAlias ($cmdletName) {
  Get-Alias |
    Where-Object -FilterScript {$_.Definition -like "$cmdletName"} |
      Format-Table -Property Definition, Name -AutoSize
}
```

### Customize your console

```
 function CustomizeConsole {
  $hostTime = (Get-ChildItem -Path $PSHOME\pwsh.exe).CreationTime
  $hostVersion="$($Host.Version.Major)`.$($Host.Version.Minor)"
  $Host.UI.RawUI.WindowTitle = "PowerShell $hostVersion ($hostTime)"
  Clear-Host
}
CustomizeConsole
```

### Add a customized PowerShell prompt

```
 function prompt {
    $Env:COMPUTERNAME + "\" + (Get-Location) + "> "
}
```

For more information about the PowerShell prompt, see about_Prompts.

For other profile examples, see Customizing your shell environment.

## The NoProfile parameter

To start PowerShell without profiles, use the **NoProfile** parameter of `pwsh.exe`, the program that starts PowerShell.

To begin, open a program that can start PowerShell, such as Cmd.exe or PowerShell itself. You can also use the Run dialog box in Windows.

Type:

```
pwsh -NoProfile
```

For a complete list of the parameters of `pwsh.exe`, type:

```
pwsh -?
```

## Profiles and execution policy

The PowerShell execution policy determines, in part, whether you can run scripts and load configuration files, including the profiles. The **Restricted** execution policy is the default. It prevents all scripts from running, including the profiles. If you use the "Restricted" policy, the profile doesn't run, and its contents aren't applied.

A `Set-ExecutionPolicy` command sets and changes your execution policy. it's one of the few commands that applies in all PowerShell sessions because the value is saved in the registry. You don't have to set it when you open the console, and you don't have to store a `Set-ExecutionPolicy` command in your profile.

## Profiles and remote sessions

PowerShell profiles aren't run automatically in remote sessions, so the commands that the profiles add aren't present in the remote session. In addition, the `$PROFILE` automatic variable isn't populated in remote sessions.

To run a profile in a session, use the Invoke-Command cmdlet.

For example, the following command runs the "Current user, Current Host" profile from the local computer in the session in `$s`.

```
Invoke-Command -Session $s -FilePath $PROFILE
```

The following command runs the "Current user, Current Host" profile from the remote computer in the session in `$s`. Because the `$PROFILE` variable isn't populated, the command uses the explicit path to the profile. We use dot sourcing operator so that the profile executes in the current scope on the remote computer and not in its own scope.

```
Invoke-Command -Session $s -ScriptBlock {
 . "$HOME\Documents\WindowsPowerShell\Microsoft.PowerShell_profile.ps1"
}
```

After running this command, the commands that the profile adds to the session are available in `$s`.

## See also

- about_Automatic_Variables
- about_Execution_Policies
- about_Functions
- about_Prompts
- about_Remote
- about_Scopes
- about_Signing
- Set-ExecutionPolicy

---