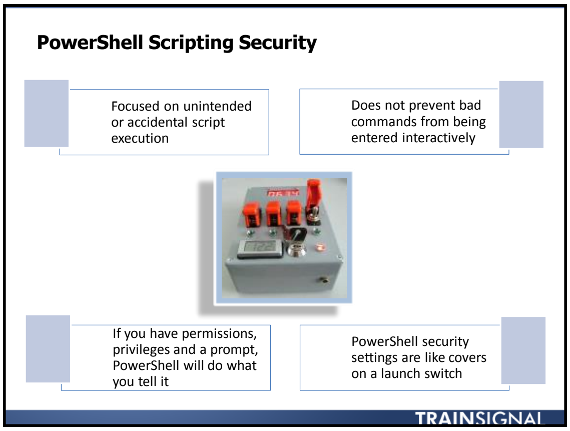
# PowerShell Scripting Basics

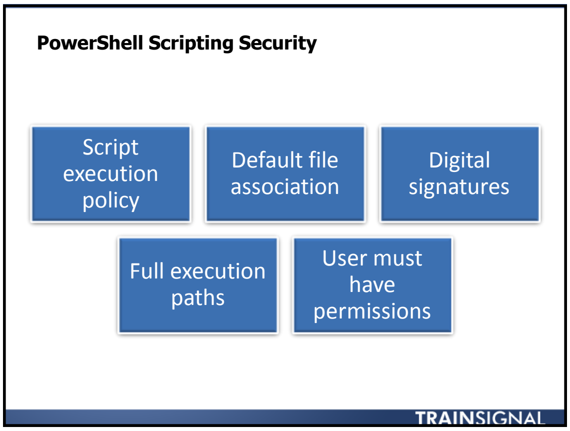
TRAINSIGNAL

THE GLOBAL LEADER IN PROFESSIONAL COMPUTER TRAINING



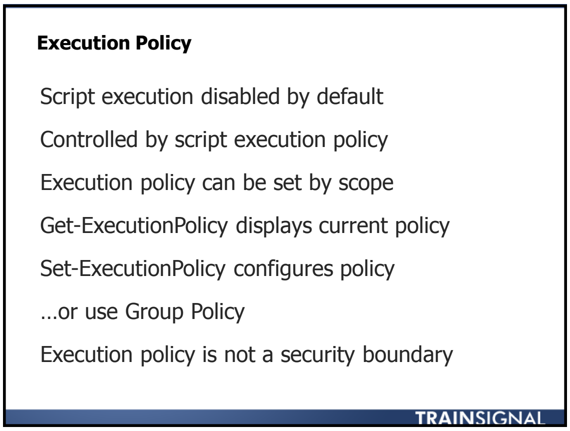
## PowerShell Scripting Security

- Focused on unintended or accidental script execution
- Does not prevent bad commands from being entered interactively
- If you have permissions, privileges and a prompt, PowerShell will do what you tell it
- PowerShell security settings are like covers on a launch switch



## PowerShell Scripting Security

- Script execution policy
- Default file association
- Digital signatures
- Full execution paths
- User must have permissions



## Execution Policy

Script execution disabled by default

Controlled by script execution policy

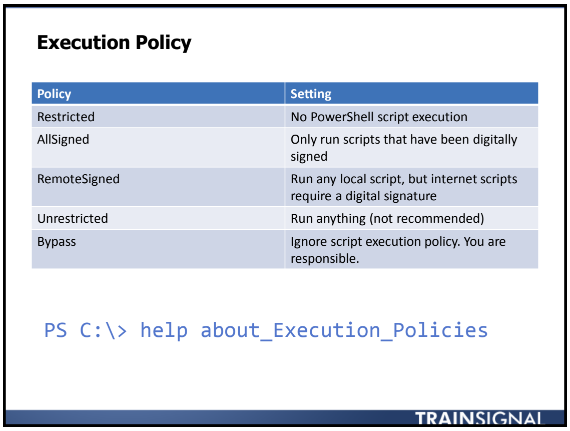
Execution policy can be set by scope

Get-ExecutionPolicy displays current policy

Set-ExecutionPolicy configures policy

…or use Group Policy

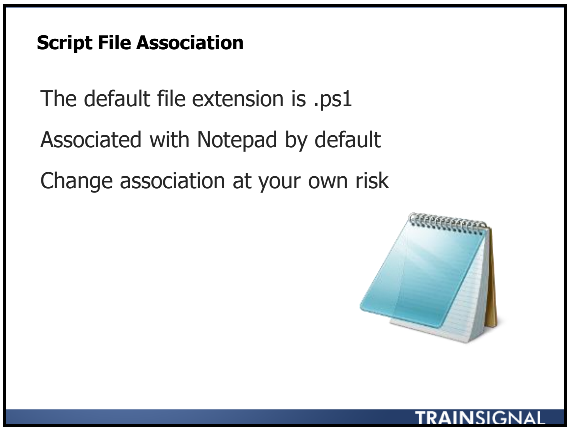
Execution policy is not a security boundary



## Execution Policy

| Policy | Setting |
|---|---|
| Restricted | No PowerShell script execution |
| AllSigned | Only run scripts that have been digitally signed |
| RemoteSigned | Run any local script, but internet scripts require a digital signature |
| Unrestricted | Run anything (not recommended) |
| Bypass | Ignore script execution policy. You are responsible. |

```
PS C:\> help about_Execution_Policies
```

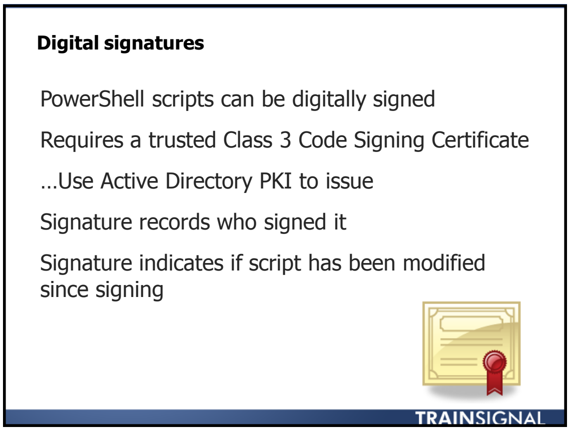


## Script File Association

The default file extension is .ps1

Associated with Notepad by default

Change association at your own risk



## Digital signatures

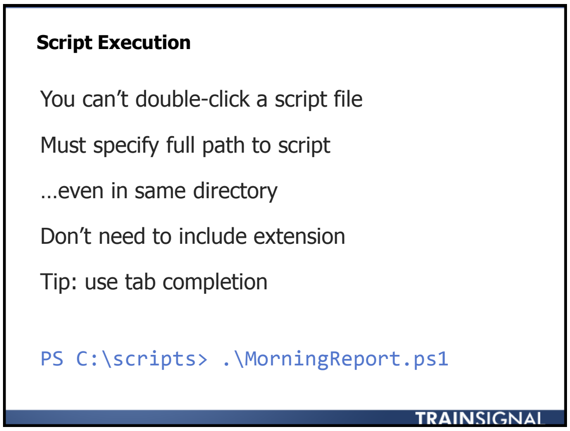
PowerShell scripts can be digitally signed

Requires a trusted Class 3 Code Signing Certificate

…Use Active Directory PKI to issue

Signature records who signed it

Signature indicates if script has been modified since signing



## Script Execution

You can’t double-click a script file

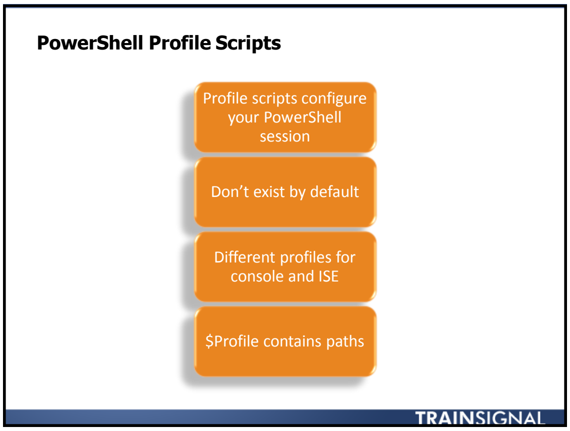
Must specify full path to script

…even in same directory

Don’t need to include extension

Tip: use tab completion

```
PS C:\scripts> .\MorningReport.ps1
```



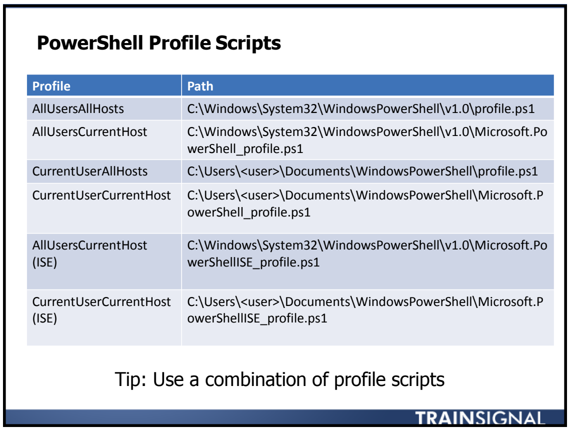
## PowerShell Profile Scripts

Profile scripts configure your PowerShell session

Don’t exist by default

Different profiles for console and ISE

$Profile contains paths

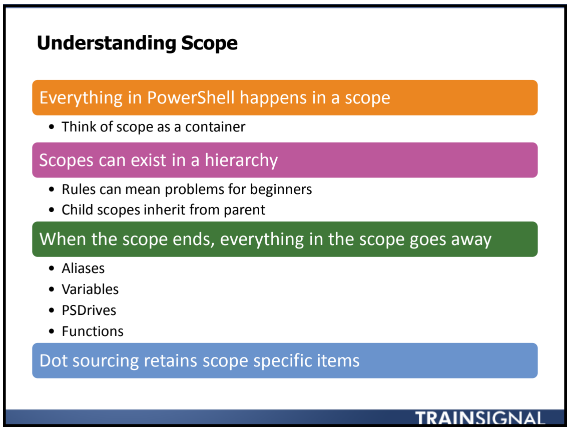


## PowerShell Profile Scripts

| Profile | Path |
|---|---|
| AllUsersAllHosts | C:\Windows\System32\WindowsPowerShell\v1.0\profile.ps1 |
| AllUsersCurrentHost | C:\Windows\System32\WindowsPowerShell\v1.0\Microsoft.PowerShell_profile.ps1 |
| CurrentUserAllHosts | C:\Users\<user>\Documents\WindowsPowerShell\profile.ps1 |
| CurrentUserCurrentHost | C:\Users\<user>\Documents\WindowsPowerShell\Microsoft.PowerShell_profile.ps1 |
| AllUsersCurrentHost (ISE) | C:\Windows\System32\WindowsPowerShell\v1.0\Microsoft.PowerShellISE_profile.ps1 |
| CurrentUserCurrentHost (ISE) | C:\Users\<user>\Documents\WindowsPowerShell\Microsoft.PowerShellISE_profile.ps1 |

Tip: Use a combination of profile scripts

## Scripting Security



## Understanding Scope

Everything in PowerShell happens in a scope

- Think of scope as a container

Scopes can exist in a hierarchy

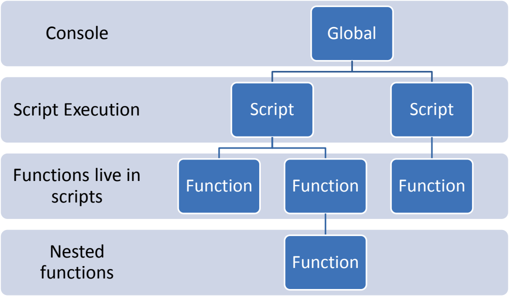
- Rules can mean problems for beginners
- Child scopes inherit from parent

When the scope ends, everything in the scope goes away

- Aliases
- Variables
- PSDrives
- Functions

Dot sourcing retains scope specific items

## PowerShell Scope



Console

Global

Script Execution

Script

Script

Functions live in scripts

Function

Function

Function

Nested functions

Function

## Scope Rules

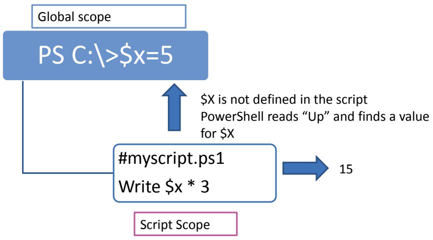
### Reading

- PowerShell searches current scope
- PowerShell searches up the scope hierarchy
- If found, uses it

### Writing

- PowerShell searches current scope
- If found, uses it
- If not found, writes to the current scope

## Understanding Scope



Global scope

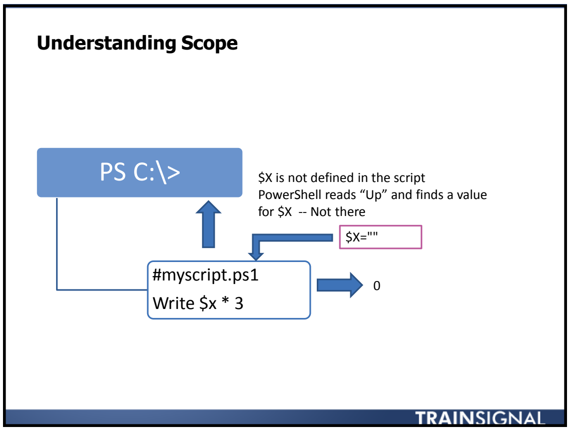
PS C:\>$x=5

$X is not defined in the script
PowerShell reads “Up” and finds a value for $X

```
#myscript.ps1
Write $x * 3
```

15

Script Scope



## Understanding Scope

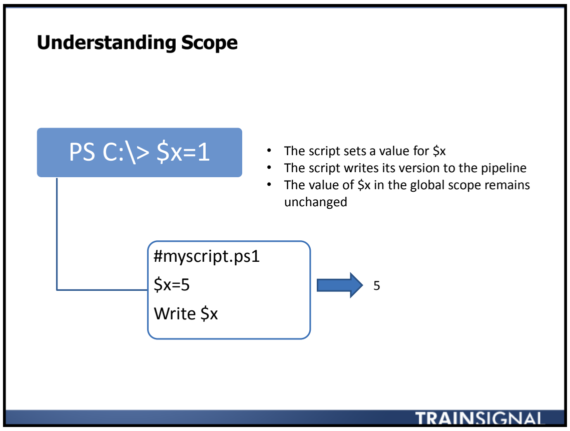
PS C:\>

$X is not defined in the script
PowerShell reads "Up" and finds a value
for $X -- Not there

$X=""

```
#myscript.ps1
Write $x * 3
```

0



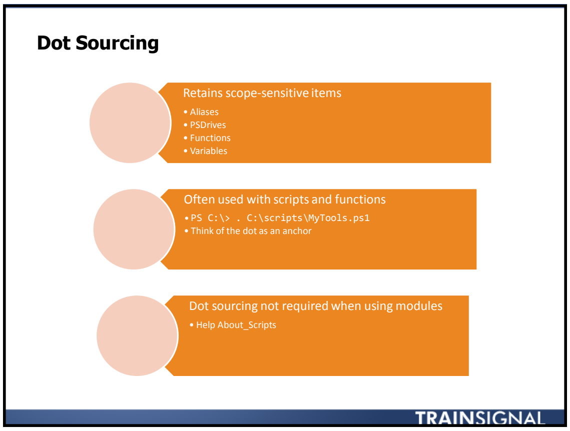
## Understanding Scope

PS C:\> $x=1

- The script sets a value for $x
- The script writes its version to the pipeline
- The value of $x in the global scope remains unchanged

```
#myscript.ps1
$x=5
Write $x
```

5



## Dot Sourcing

Retains scope-sensitive items

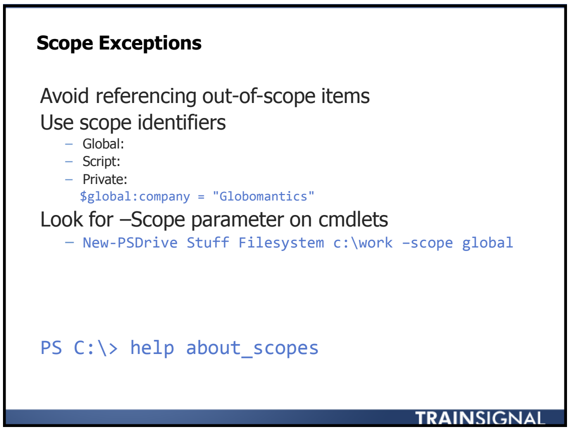
- Aliases
- PSDrives
- Functions
- Variables

Often used with scripts and functions

- `PS C:\> . C:\scripts\MyTools.ps1`
- Think of the dot as an anchor

Dot sourcing not required when using modules

- Help About_Scripts



## Scope Exceptions

Avoid referencing out-of-scope items

Use scope identifiers

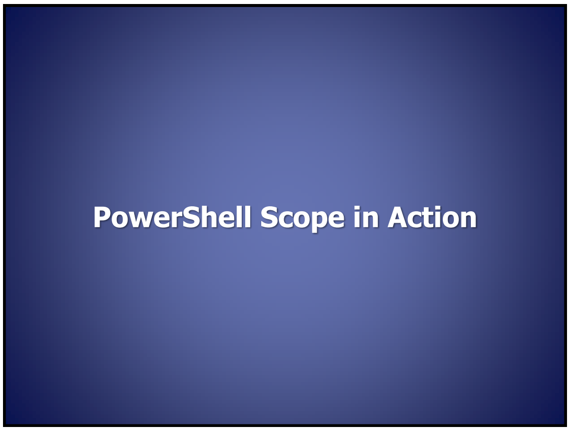
- Global:
- Script:
- Private:

```
$global:company = "Globomantics"
```

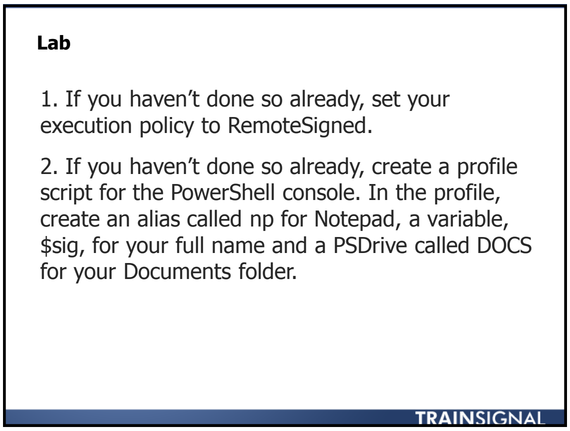
Look for –Scope parameter on cmdlets

```
– New-PSDrive Stuff Filesystem c:\work –scope global
```

```
PS C:\> help about_scopes
```



## PowerShell Scope in Action



## Lab

1. If you haven't done so already, set your execution policy to RemoteSigned.

2. If you haven't done so already, create a profile script for the PowerShell console. In the profile, create an alias called np for Notepad, a variable, $sig, for your full name and a PSDrive called DOCS for your Documents folder.