PowerShell Script Error Handling & Debugging

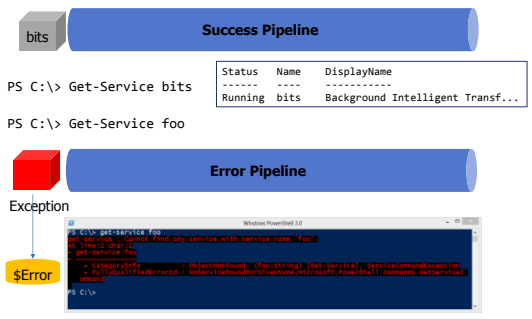
TRAINSIGNAL

THE GLOBAL LEADER IN PROFESSIONAL COMPUTER TRAINING™

Understanding Exceptions and Errors

- In PowerShell the *error* is the message you see when there is a problem
- An *exception* is the object that contains the error
- Exceptions stored in $Error automatic variable
- Default is 256 exceptions ($MaximumErrorCount)

The Error Pipeline



ErrorActionPreference

Control behavior of the error pipeline

Set at scope level: $ErrorActionPreference

Set at the cmdlet level with –ErrorAction parameter (-ea)

Preferences

- SilentlyContinue (0)
- Stop (1)
- Continue (2) [this is the default behavior]
- Inquire (3)
- Ignore (4) [parameter value only]

Suppressing the pipeline doesn't stop the exception captured in $Error

- Unless you Ignore

Terminating vs Non-Terminating Exceptions

PowerShell has two types of exceptions:

Terminating

- PowerShell terminates the pipeline

Non-Terminating

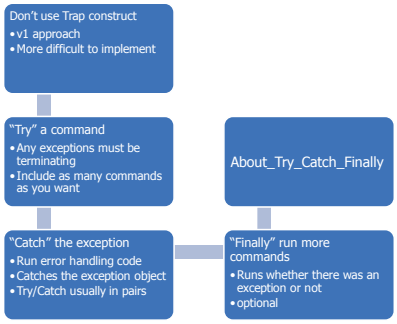
- PowerShell tries to keep going

You can only "handle" terminating exceptions

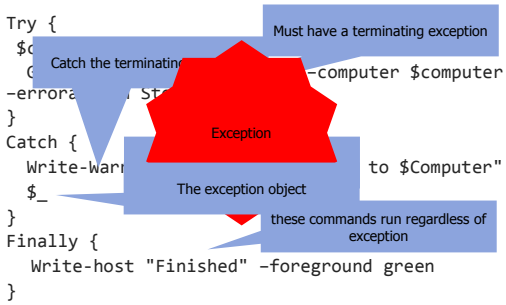
Force exceptions to be terminating

```
$ErrorActionPreference = Stop
-erroraction Stop
```


Error Handling With Try/Catch/Finally



Try/Catch



Throwing Your Own Exceptions

Write-Error

- Writes a non-terminating exception
- Write your own exception object

Throw

- Write a terminating exception
- Help about_throw

Error Handling Demo

Debugging Your Script

Debugging is determining where reality diverges from expectation

Reduce bug opportunities from the beginning

- Write your scripts in a scripting editor
- …or at least the PowerShell ISE
- Layout your script
- Use Set-StrictMode

```
PS C:\> Help about_debuggers
```


Debugging Your Script

Use Write-Debug messages	Use Write-Verbose messages	Insert breakpoints
Enable debug pipeline • -debug • $DebugPreference	Enable Verbose pipeline • -verbose • $VerbosePreference	Enter debug mode • Run in console or ISE

Set-StrictMode

Many bugs are result of typos

Use Set-StrictMode to enforce proper coding

Set-StrictMode is scope-specific

Set –version to PowerShell version

PowerShell will throw an exception if there is a violation

StrictMode Version

Version	Effect
1.0	Prohibits references to uninitialized variables, except for uninitialized variables in strings.
2.0	Prohibits references to uninitialized variables (including uninitialized variables in strings). Prohibits references to non-existent properties of an object. Prohibits function calls that use the syntax for calling methods. Prohibits a variable without a name (${}).
Latest	Get whatever the latest restrictions might be. Future-proofs your scripts.

Set-StrictMode ensures you don't reference variables or objects that haven't been defined in the scope

Script Debugging Demo

Best Practices

- Do not turn off the error pipeline
- Include error handling from the very beginning of your scripts and functions
- Write and test in small steps
- Use Set-StrictMode
- Do not write PowerShell scripts in Notepad

Lab

No lab for this lesson. Read the "about" help topics on error handling, try/catch and debugging.