PowerShell for .Net Devs

All the awesome little things you didn’t know you could do.

What’s it good for anyway?

- Anything REMOTE
- Anything you can do with .Net
- Anything you want to automate
- Anything you would do with a Console Application.
- Examples
  - Remote Deployment / Continuous Integration (TeamCity plays nice)
  - File system management
  - IIS management
  - Windows services management
  - MSBuild - Pre/Post Build Events
  - Nuget - Install/Uninstall

But I Suck at PowerShell

How to Not Suck at PowerShell

1. Know what it CAN do
2. Keep it in mind day to day
3. Google the funky syntax when you need it.
4. Save tons of time
5. Beg your DM for more billable hours
6. Enjoy a beer - Sierra Idle

As long as you can do it in .Net, with a little research, you can figure out how to do it in PowerShell.

Tools

- PowerShell IDE
  - Intellisense!
  - Debugging!
- Visual Studio
  - TextHighlighterExtension2013 (or 2012, or 2010)
    - *Also handy for .bat files and lots of other stuff*
  - No intellisense :(
  - No debugging :(
  - Don’t develop here unless it’s a minor tweak.

Security Levels

Scripts aren't allowed to run by default.

Set-ExecutionPolicy RemoteSigned

GOTCHA: Be aware if you are running x64 or x86.

(Visual Studio pre/post build and Nuget installs run out of x86)

- **Restricted**: Does not load configuration files or run scripts. "Restricted" is the default execution policy.
- **AllSigned**: Requires that all scripts and configuration files be signed by a trusted publisher, including scripts that you write on the local computer.
- **RemoteSigned**: Requires that all scripts and configuration files downloaded from the Internet be signed by a trusted publisher.
- **Unrestricted**: Loads all configuration files and runs all scripts. If you run an unsigned script that was downloaded from the Internet, you are prompted for permission before it runs.
- **Bypass**: Nothing is blocked and there are no warnings or prompts.
- **Undefined**: Removes the currently assigned execution policy from the current scope. This parameter will not remove an execution policy that is set in a Group Policy scope.

The Basics - Terms & Syntax

- Variables, Operators & Cmdlets
- Script Blocks
- Pipelines
- Loading & Using .dll's
- Runtime Classes
- Scripts, Modules & Snapins OH MY!
  - Script.ps1 = quick and dirty executable file
  - Module.psm1 = reusable file loaded by other scripts
  - Snapin = deployable/installable file for the outside world
    - Snapins are not developed in PowerShell
    - Developed in .Net (special type of visual studio project)
    - Extend CustomPSSnapin from System.Management.Automation

The Basics - PSDrives

- Everything is a drive
  - File System
  - Registry
  - IIS
  - Any collection from which you create a PSDrive

Remoting!!!

Can only remote to a machine with the WinRM service running

Enable-PSRemoting

GOTCHA: Manage & CLOSE your sessions.

The target machine can only have 5 by default

GOTCHA: No closures (yet… it's on the horizon).

It's running on another machine. Any modules or .dll's you loaded aren't there anymore

Manually:

```
Enter-PSSession -ComputerName $theTargetMachine
Get-ChildItem -Path "c:/"
Exit-PSSession
```

In a Script

```
$sess = Get-PSSession -ComputerName $theTargetMachine
Invoke-Command -Session $sess -ScriptBlock { Get-ChildItem -Path "c:/" }
Remove-PSSession -Session $sess
```

IIS Management

- WebAdministration Module
  - Creates a PSDrive called IIS:\
  - Drive structure mirrors IIS and web.config XML within each site
  - Performance kind of sucks
  - Good for small simple tasks
  - XPath queries and XML modification for advanced changes
  - x64 only
- System.Web.Administration.ServerManager
  - Much faster
  - Directly accessing .Net classes means easier to make advanced changes
  - x86 compatible (MSBuild Pre/Post build & Nuget)

Windows Service Management

- Handy Cmdlets
  - Start-Service
  - Stop-Service
  - New-Service
- Accessing the WMI Service Object
  - $service = Get-WmiObject -Class Win32_Service -Filter "Name = 'My Service'"
- Any cmd tool for managing services can be used programmatically through PowerShell as well

Tangent: *Highly recommend developing services with TopShelf*

Runs as both a console app or a service, awesome for debugging, easy to deploy. Oh and hotswapping .dll's is a cool feature too.

MSBuild - Pre/Post Build

MSBuild runs cmd.exe (x86), from which we execute powershell.exe

Either Through the Project's Properties Editor Pre/Post Build Field:

```
powershell.exe -File $(ProjectDir)\test.ps1 -TargetDir $(TargetDir) -Configuration $(Configuration)
```

Or Edit the .csproj XML File Directly:

```
<PropertyGroup>
    <PreBuildEvent>powershell.exe -File $(ProjectDir)\test.ps1 -TargetDir $(TargetDir) -Configuration $(Configuration)
</PreBuildEvent>
</PropertyGroup>
```

Test.ps1 (located in root of project):

```
param(
    $TargetDir,
    $Configuration
)
#This will show up in the build output window
Write-Host "The bin is at $TargetDir and the build configuration is $Configuration"
```

Nuget

Automatically Running PowerShell Scripts During Package Installation and Removal

A package can include PowerShell scripts that automatically run when the package is installed or removed. NuGet automatically runs scripts based on their file names using the following conventions:

- **Init.ps1** runs the first time a package is installed in a solution.
  - If the same package is installed into additional projects in the solution, the script is not run during those installations.
  - The script also runs every time the solution is opened. For example, if you install a package, close Visual Studio, and then start Visual Studio and open the solution, the Init.ps1 script runs again.
- **Install.ps1** runs when a package is installed in a project.
  - If the same package is installed in multiple projects in a solution, the script runs each time the package is installed.
  - The package must have files in the content or lib folder for Install.ps1 to run. Just having something in the tools folder will not kick this off.
  - If your package also has an init.ps1, install.ps1 runs after init.ps1.
- **Uninstall.ps1** runs every time a package is uninstalled.
- These **files should be located in the tools directory** of your package.
- At the top of your file, add this line: **param($installPath, $toolsPath, $package, $project)**
  - $installPath is the path to the folder where the package is installed
  - $toolsPath is the path to the tools directory in the folder where the package is installed
  - $package is a reference to the package object.
  - $project is a reference to the EnvDTE project object and represents the project the package is installed into. Note: This will be null in Init.ps1. In that case doesn't have a reference to a particular project because it runs at the solution level. The properties of this object are defined in the MSDN documentation.
- When you are testing $project in the console while creating your scripts, you can set it to $project = Get-Project

Automate Tedious Tasks!

If you ever consider spinning up a quick & dirty Console Application

DO POWERSHELL INSTEAD!!!

Why?

- No need to recompile every time you make a change.
- Only need to add 1 file to source control
  - (instead of the .exe and source code it came from)

Examples

- Shuffling or renaming folders & files
- generate XML map of file structure
- generate JSON from .csv
- Mapping any annoying data format to any other annoying data format

Ain’t Learnt it Good? More Learnin’

See slide 2. You don’t really need to learn it, just Google a lot ;)

We’re talking quick and dirty utilities here, not robust applications, just get in there and mess around.*

MSDN
http://msdn.microsoft.com/en-us/library/dd835506(v=vs.85).aspx

Tutorials
http://www.powershellpro.com/powershell-tutorial-introduction/

Michael Evans is not to be held liable when you blow up your computer. Avoid the command: Explode-Computer -Force