PowerShell Flow Control

SCRIPTING ESSENTIALS

DR. BURKMAN

PowerShell Basics

PowerShell

We will use 64 bit PS and PS ISE (not x86)

$PSVersionTable

PowerShell.exe is the text command window

PowerShell ISE is the scripting environment

- Right click the icon as run as administrator

Increase font with ctrl + mouse wheel or with slider in lower right of screen

ISE is really where we want to be to build scripts

Running this at home

If you run PowerShell on your own box you may find that you can't run scripts.

- Get-ExecutionPolicy
- Set-ExecutionPolicy unrestricted

Obviously be smart about this. You just opened your machine to run any PowerShell code from any source, and there are a lot of modern hacks using PowerShell. So either do this on a junk machine (or play VM) or restrict the policy when done with coursework

Windows PowerShell has four different execution policies:

- Restricted - No scripts can be run. Windows PowerShell can be used only in interactive mode.
- AllSigned - Only scripts signed by a trusted publisher can be run.
- RemoteSigned - Downloaded scripts must be signed by a trusted publisher before they can be run.
- Unrestricted - No restrictions; all scripts can be run.
  - From: https://www.mssqltips.com/sqlservertip/2702/setting-the-powershell-execution-policy/

ISE

Feels a bit like IDLE

Script Editor pane

Console pane

Commands Explorer

Tab auto-complete

- In console, type Get-S and hit tab a few times
  - Both autocompletion and Intellisense (the menu)
- Try dir C:\ and tabs

Up arrow for previous commands

ISE

Update help with Update-Help
- This is essential
- You must be Administrator

Invoke help with help or man
- Help while
- Man while

Clear
- Clears the console

Help *log* or help *event* to get a list of possible relevant help topics

Get-Command *firewall* to get PowerShell (and windows) commands

Try: help about

ISE

Get-EventLog system

- Get-EventLog –List
- Get-EventLog Application > C:\Users\Burkman\Desktop\events.txt
  - > and >> work just like in BASH

cmdlets are verb/noun pairs

Can create a transcript of everything you do in PS session

- Start-Transcript
- Stop-Transcript

ISE

You have access to the .net built in classes like Math.

- $dog = [math]::abs($cat)

Can use piping

- Get-Process | Where-Object { -not $_.Responding }
- Get-Process | Where-Object PriorityClass –eq “Normal”
- Get-Content "C:\Users\Burkman\Desktop\events.txt " | select -First 10 | Out-File "C:\Users\Burkman\Desktop\toptenevents.txt"

PowerShell Flow Control

PowerShell Flow Control

- Some basics:
  - How variables are denoted (hint: about_variables)
    - Put a $ in front of the variable name
  - How to pause

```
Read-Host -Prompt "Press enter to close the program"
```

  - Break and continue as normal
  - $true and $false for booleans
  - + and * work just like in Python

PowerShell Flow Control

Comments:

```
#Comment one line

<# comment
over several
lines #>
```

PowerShell Flow Control

User input and printing to console. Use Write-Host for printing to the console:

```
$dog = Read-Host -Prompt "Please enter a word"
Write-Host $dog
Write-Host "This is $dog"

#There is also Write-Output, which allows the output to be redirected

Write-Output $dog > C:\Users\Burkman\Desktop\dog.txt
```

Quotes

Single quotes are literal, double quotes allow for variable unpacking:

```
$cat = 5

Write-Output "$cat is five"

Write-Output '$cat is five'
```

```
PS C:\WINDOWS\system32> C:\Users\Burkman\Desktop\jim.ps1
5 is five
$cat is five
```

PowerShell Flow Control

Getting random values

```
clear

$myRan = Get-Random -Minimum 10 -Maximum 20

Write-Host $myRan
Read-Host -Prompt "Press any key to continue"
```

This will only get up to 19. The minimum is included, the maximum is not.

Note

Examples throughout these slides are excerpts from the Windows PowerShell Cookbook

Nice links:

- https://akr.am/blog/posts/a-python-developers-guide-to-powershell
  - Note: his PS XKCD example is broken
- http://blogs.lessthandot.com/index.php/datamgmt/dbprogramming/a-cheat-sheet-for-all/

PowerShell is C# - ish, so think in term of braces and you'll be fine

https://github.com/PoshCode/PowerShellPracticeAndStyle/issues/81

Allman style

The Allman style is named after Eric Allman, who wrote many of the BSD utilities. It puts braces on their own lines, indented to the same level as the control statement, and indents statements within the braces.

This style is the Visual Studio default indenting style for C# and is the standard for dotnet, PowerShell, asp.net, and basically, all Microsoft C# projects.

```
enum Color
{
    Black,
    White
}

function Test-Code
{
    [CmdletBinding()]
    param
    (
        [int]$ParameterOne
    )
    end
    {
        if(10 -gt $ParameterOne)
        {
            "Greater"
        }
        else
        {
            "Lesser"
        }
    }
}
```

Bash-like Comparisons

Comparison operators
-eq , -ne , -ge , -gt , -in , -notin , -lt , -le , -like , -notlike , -match , -notmatch ,-contains , -notcontains , -is , -isnot (put c in front to make case sensitive, like –ceq)

Logical operators -and , -or , -xor , -not

If statement

if, elseif, and else Statements

```
if(condition)
{
statement block
}
elseif(condition)
{
statement block
}
else
{
statement block
}
```

IF Example (with While)

```
clear #This clears the output screen

while ($true) #while works like we already know with break and continue
{
    $ans = Read-Host -Prompt "Enter an integer"

    if ($ans/1 -gt 10) #$ans/1 forces the input to be an integer
    {
        write-host "big"
    }
    else
    {
        Write-host "small"
    }
}
```

Try/Catch

```
while ($true)
{
    $ans = Read-Host "Enter a number"
    try
    {
        $ans -eq $ans/1 | Out-Null
    }
    catch
    {
        Write-host "oops"
        continue
    }
    if ($ans/1 -is [int])
    {
        write $ans
        break
    }
    else
    {
        write-host "$ans is not an integer"
        continue
    }
}
```

For Statement

help about_For

for (<init>; <condition>; <repeat>)
{<statement list>}

```
for($i=1; $i -le 10; $i++)
{
    Write-Host $i
}
```

For Statement

```
clear

for ($i = 30; $i -gt 10; $i--)
{
    Write-Host $i
}

#This is like looping through a range in Python
```

Foreach Statement

Help about_Foreach

```
$dog=(1,2,3,'a','b','c')

foreach ($cat in $dog)
{
    Write-Host($cat)
}
```

Helpful Miscellanea

Backtick is used to escape. That's above the tab key on most keyboards.

Outputting text with an additional line feed:

```
Write-host("Welcome to the Favorite Zone `r`n")
```

You can exit the script with exit.