PowerShell

CSCI 4417/5417

Introduction to System Administration

East Tennessee State University
Department of Computing
Jack Ramsey, Lecturer

Where are we going?

What is Windows PowerShell?

Where is it available?

What can it be used for?

Why is it good for system admins?

PowerShell

Rich scripting language for Microsoft systems

Can access

- Command line utilities (ipconfig, dir, etc)
- Windows Management Instrumentation (WMI)
- VBscripts

Used to automate tasks and otherwise improve the admins tasks

PowerShell

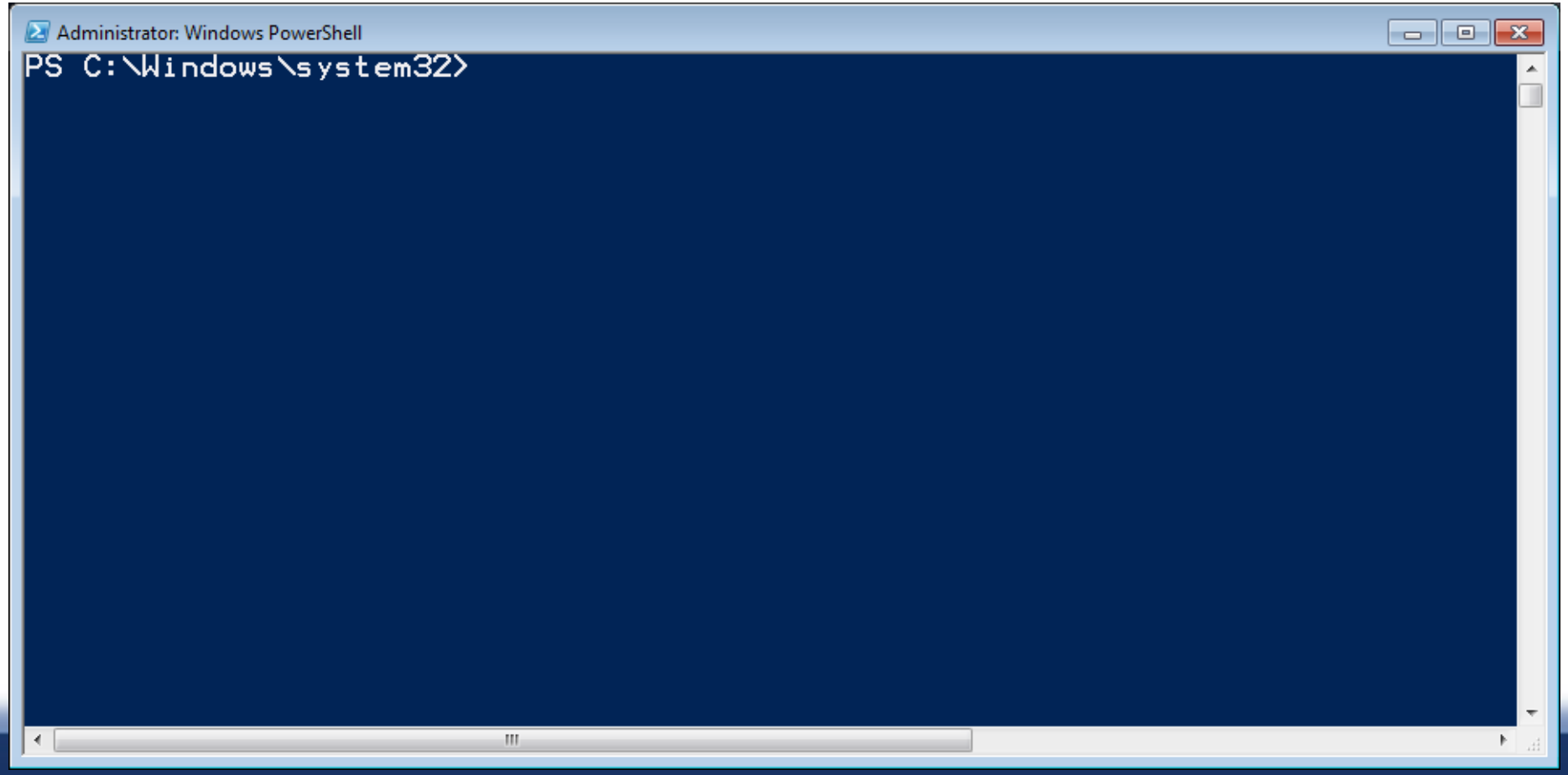
Object-based shell

Automation & management engine

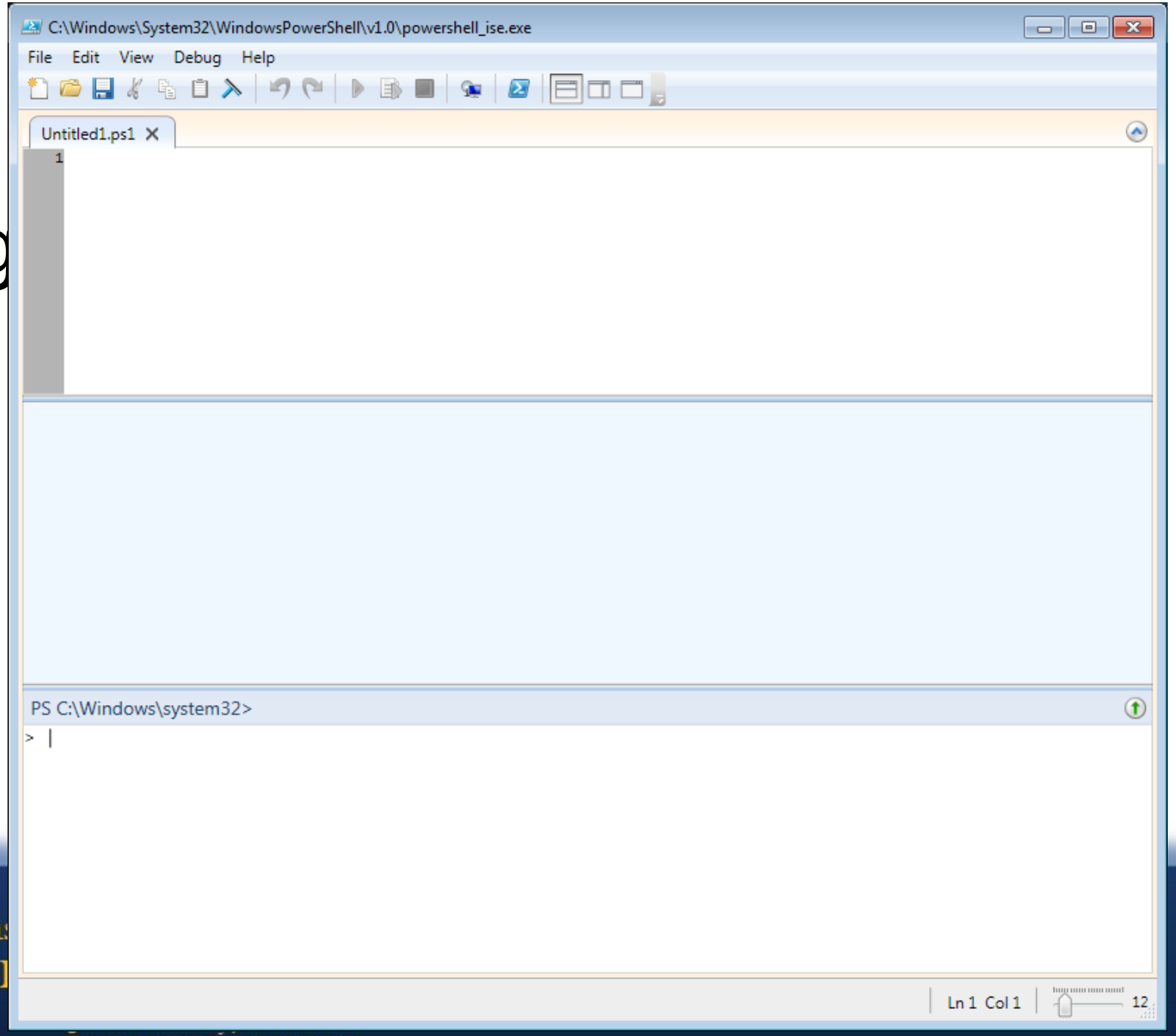
PowerShell host v. PowerShell engine

- Give commands to engine via host
- Engine is a set of .NET Framework classes stored in a DLL file
- We don’t interact with the engine directly

PowerShell Console



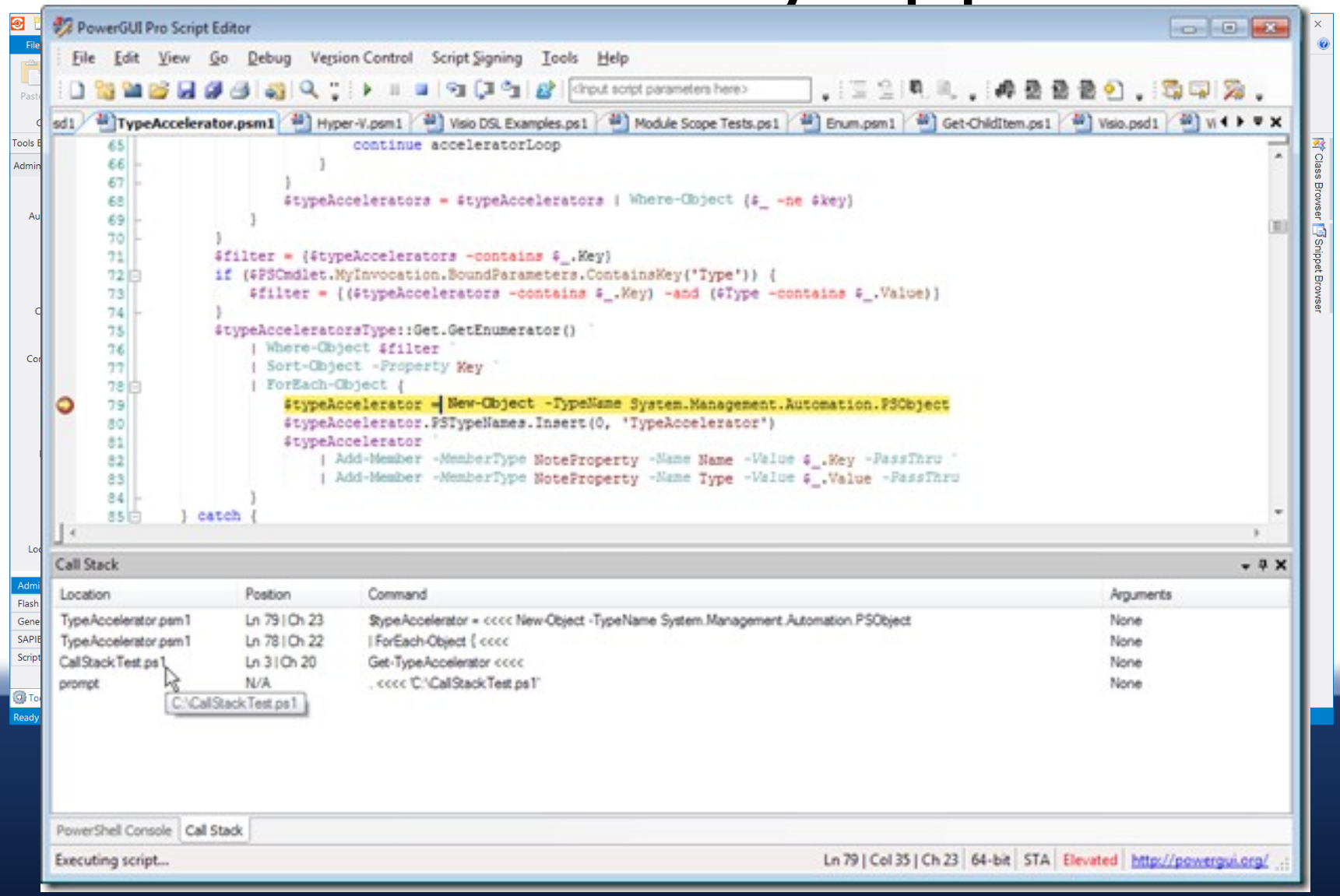
PowerShell Integrated Scripting Editor (ISE)



PowerShell - Third Party Apps

PowerGUI

PowerShell Pro Script



ETSU

Versions - Which One?

To date, there are 5

Bundled with latest versions of Windows

Earlier versions of Windows (e.g., XP) - had to download

Updates available via Windows Management Foundation (WMF)

Versions

Administrator: Windows PowerShell

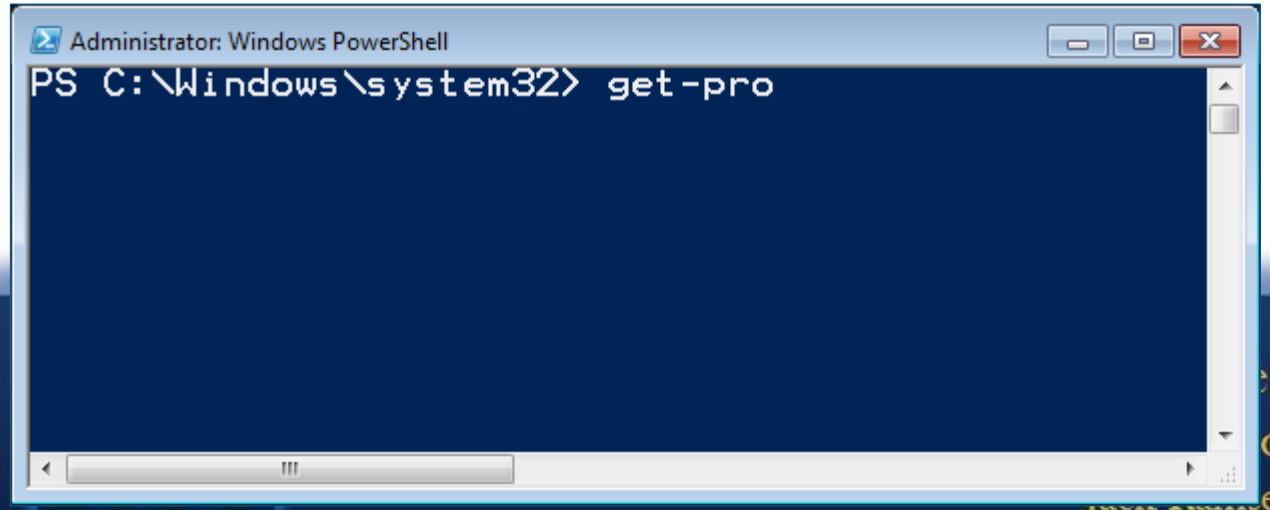
```
PS C:\Windows\system32> $PSVersionTable

Name                           Value
----                           -----
CLRVersion                     2.0.50727.5485
BuildVersion                   6.1.7601.17514
PSVersion                      2.0
WSManStackVersion              2.0
PSCompatibleVersions           {1.0, 2.0}
SerializationVersion           1.1.0.1
PSRemotingProtocolVersion      2.1


PS C:\Windows\system32>
```

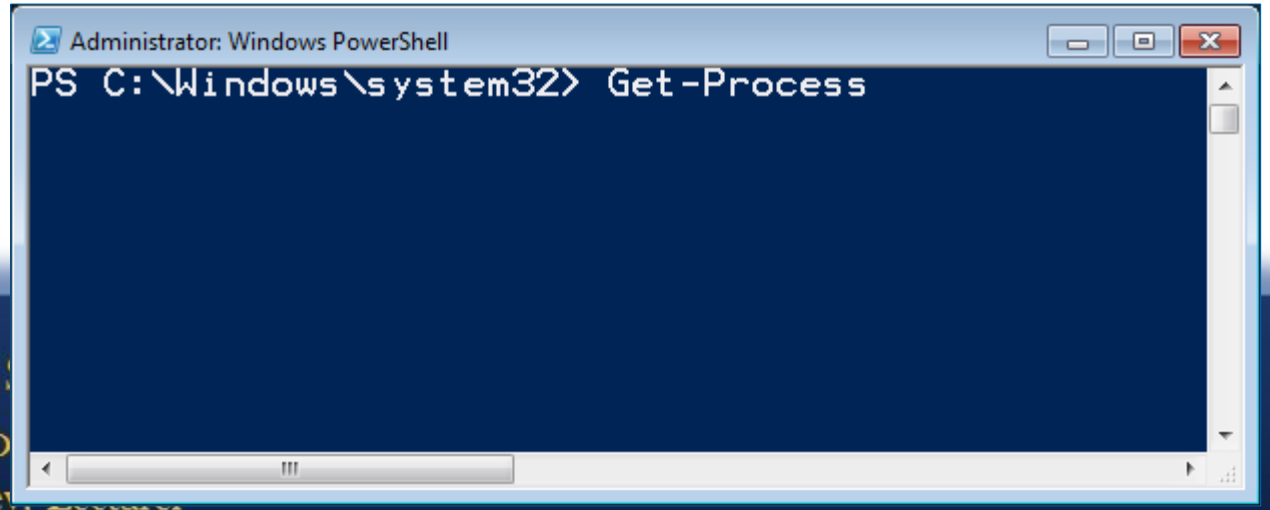
TAB-Completion / Command History

If you type in part of a command, typing the TAB key will complete the pattern

If more than one pattern match, repeated TABs will scroll through the commands that match

Up and down arrows will scroll through recent commands





Transcripts

In console (not ISE), **`Start-Transcipt`** records every command entered, along with its output and errors to a text file

```
Start-Transcript c:\work\Monday.txt
```

`Stop-Transcript` stops the recording

Simple PowerShell Example (1)

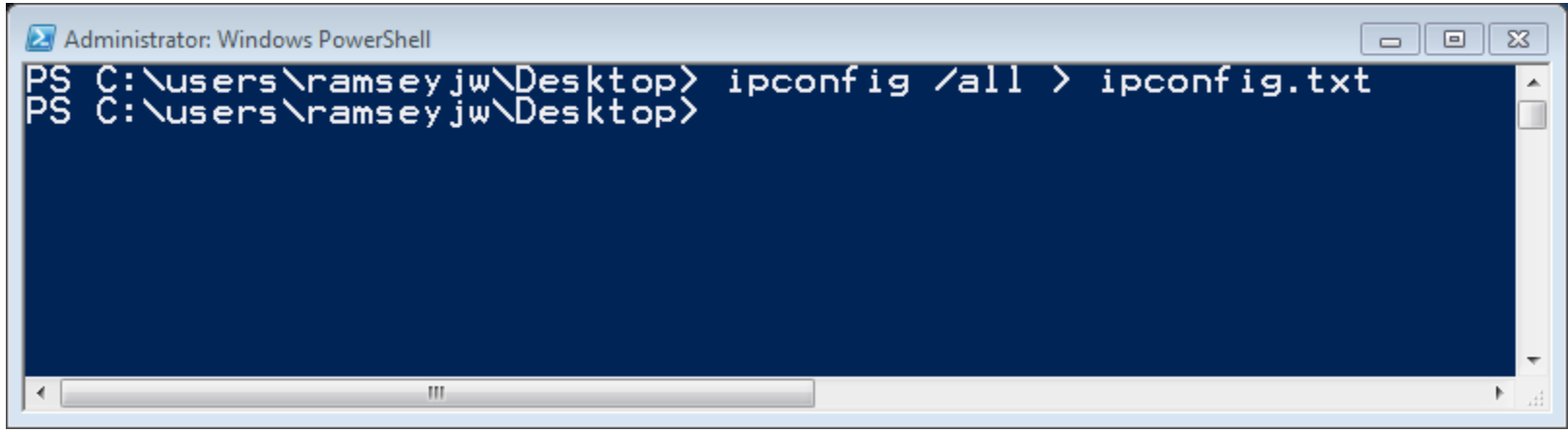
Open PowerShell by Start | Run | Windows PowerShell

```
ipconfig /all
```

Dump output into a file

```
ipconfig /all >
ipconfig.txt
```

Open file in notepad



Simple PowerSh
Example (1)

C:\Users\ramseyjw\Desktop\ipconfig.txt - Notepad++

File Edit Search View Encoding Language Settings Macro Run Plugins Window ?

midterm.html | resume.html | bootstrap.css | bootstrap.min.css | authlog.txt | template.html | index.html | ipconfig.txt

```
Windows IP Configuration

   Host Name . . . . . . . . . . . . : ETSU906444
   Primary Dns Suffix  . . . . . . . : etsu.edu
   Node Type . . . . . . . . . . . . : Hybrid
   IP Routing Enabled. . . . . . . . : No
   WINS Proxy Enabled. . . . . . . . : No
   DNS Suffix Search List. . . . . . : etsu.edu

Ethernet adapter NETGEAR-VPN:

   Connection-specific DNS Suffix  . :
   Description . . . . . . . . . . . : TAP-Windows Adapter V9
   Physical Address. . . . . . . . . : 00-FF-C7-1F-B5-D0
   DHCP Enabled. . . . . . . . . . . : Yes
   Autoconfiguration Enabled . . . . : Yes
   IPv4 Address. . . . . . . . . . . : 192.168.1.9(Preferred)
   Subnet Mask . . . . . . . . . . . : 255.255.255.0
   Lease Obtained. . . . . . . . . . : Monday, October 26, 2015 7:38:48 AM
   Lease Expires . . . . . . . . . . : Friday, November 06, 2015 7:43:18 AM
   Default Gateway . . . . . . . . . :
   DHCP Server . . . . . . . . . . . : 192.168.1.1
   DNS Servers . . . . . . . . . . . : 192.168.1.1
   NetBIOS over Tcpip. . . . . . . . : Enabled

Ethernet adapter Local Area Connection:

   Connection-specific DNS Suffix  . : ETSU.EDU
   Description . . . . . . . . . . . : Intel(R) 82579LM Gigabit Network Connection
   Physical Address. . . . . . . . . : 18-03-73-E2-5E-38
```

Normal text file | length : 3040 lines : 68 | Ln : 1 Col : 1 Sel : 0 | 0 | Dos\Windows | UCS-2 LE BOM | INS

Simple PowerShell Example (2)

Put multiple commands on a line and let OS handle timing and synchronization of output

Separate with a semicolon – ‘;’

Use a backtick (`) for multi-line commands

```
ipconfig /all > tshoot.txt; route print >>
tshoot.txt; netstat >> tshoot.txt; net
statistics workstation >> tshoot.txt
```

Simple PowerShell Example (2)

Administrator: Windows PowerShell

```
PS C:\users\ramseyjw\Desktop> ipconfig /all > tshoot.txt; `
>> route print >> tshoot.txt; `
>> netstat >> tshoot.txt; `
>> net statistics workstation >> tshoot.txt;
>>
PS C:\users\ramseyjw\Desktop>
```

```
===========================================================================
Interface List
 17...00 ff 5e 45 45 5b ......TAP-Windows Adapter V9
 11...64 00 6a 83 eb 3b ......Intel(R) Ethernet Connection I217-LM
 13...0a 00 27 00 00 00 ......VirtualBox Host-Only Ethernet Adapter
 15...00 50 56 c0 00 01 ......VMware Virtual Ethernet Adapter for VMnet1
 16...00 50 56 c0 00 08 ......VMware Virtual Ethernet Adapter for VMnet8
  1...........................Software Loopback Interface 1
===========================================================================

IPv4 Route Table
===========================================================================
Active Routes:
Network Destination        Netmask          Gateway       Interface  Metric
          0.0.0.0          0.0.0.0      151.141.90.1    151.141.91.21     20
        127.0.0.0        255.0.0.0         On-link         127.0.0.1    306
        127.0.0.1  255.255.255.255         On-link         127.0.0.1    306
  127.255.255.255  255.255.255.255         On-link         127.0.0.1    306
     151.141.90.0    255.255.254.0         On-link     151.141.91.21    276
    151.141.91.21  255.255.255.255         On-link     151.141.91.21    276
   151.141.91.255  255.255.255.255         On-link     151.141.91.21    276
      192.168.1.0    255.255.255.0         On-link       192.168.1.8    276
      192.168.1.8  255.255.255.255         On-link       192.168.1.8    276
    192.168.1.255  255.255.255.255         On-link       192.168.1.8    276
     192.168.56.0    255.255.255.0         On-link      192.168.56.1    266
     192.168.56.1  255.255.255.255         On-link      192.168.56.1    266
   192.168.56.255  255.255.255.255         On-link      192.168.56.1    266
     192.168.79.0    255.255.255.0         On-link      192.168.79.1    276
     192.168.79.1  255.255.255.255         On-link      192.168.79.1    276
   192.168.79.255  255.255.255.255         On-link      192.168.79.1    276
    192.168.188.0    255.255.255.0         On-link     192.168.188.1    276
    192.168.188.1  255.255.255.255         On-link     192.168.188.1    276
  192.168.188.255  255.255.255.255         On-link     192.168.188.1    276
        224.0.0.0        240.0.0.0         On-link         127.0.0.1    306
        224.0.0.0        240.0.0.0         On-link     151.141.91.21    276
        224.0.0.0        240.0.0.0         On-link      192.168.56.1    266
        224.0.0.0        240.0.0.0         On-link       192.168.1.8    276
        224.0.0.0        240.0.0.0         On-link      192.168.79.1    276
        224.0.0.0        240.0.0.0         On-link     192.168.188.1    276
  255.255.255.255  255.255.255.255         On-link         127.0.0.1    306
  255.255.255.255  255.255.255.255         On-link     151.141.91.21    276
  255.255.255.255  255.255.255.255         On-link      192.168.56.1    266
  255.255.255.255  255.255.255.255         On-link       192.168.1.8    276
  255.255.255.255  255.255.255.255         On-link      192.168.79.1    276
  255.255.255.255  255.255.255.255         On-link     192.168.188.1    276
===========================================================================
Persistent Routes:
  None

IPv6 Route Table
===========================================================================
Active Routes:
 If Metric Network Destination      Gateway
  1    306 ::1/128                  On-link
  1    306 ff00::/8                 On-link
===========================================================================
Persistent Routes:
  None
```

East Tennessee St
Department of C
Jack Ramsey, I

Cmdlets

“lightweight commands” that return .NET Framework objects

- Mini programs

Cmdlets are instances of .NET Framework classes; they are not stand-alone executables

Cmdlets can be created from as few as a dozen lines of code

100s of built-in Cmdlets and we can write our own

whatif & confirm

whatif - Used to test what a command will do without executing it

confirm - Prompts user to confirm execution

Administrator: Windows PowerShell

```
PS C:\users\ramseyjw\Desktop> notepad
PS C:\users\ramseyjw\Desktop> Get-Process notepad

Handles  NPM(K)    PM(K)      WS(K) VM(M)   CPU(s)     Id ProcessName
-------  ------    -----      ----- -----   ------     -- -----------
     65       7     1932       6452    75     0.02   2608 notepad


PS C:\users\ramseyjw\Desktop> Stop-Process -Id 2608 -whatif
What if: Performing operation "Stop-Process" on Target "notepad (2608)".
PS C:\users\ramseyjw\Desktop> Stop-Process -Id 2608 -confirm

Confirm
Are you sure you want to perform this action?
Performing operation "Stop-Process" on Target "notepad (2608)".
[Y] Yes  [A] Yes to All  [N] No  [L] No to All  [S] Suspend  [?] Help
(default is "Y"):
```

Installing Help Files

PowerShell does not ship with help files

```
Update-Help -Module * -Force -ea 0
```

`-ea 0` suppresses errors that are triggered by unsupported modules. We can ignore these errors.

Get-Help

Works like man pages on Linux

Core Cmdlet's Help Topics

http://technet.microsoft.com/en-us/library/jj583014.aspx (v2 & v3)

Example get-date

To get help: `get-help get-date`

Administrator: Windows PowerShell

```
PS C:\users\ramseyjw\Desktop> get-date
Thursday, November 05, 2015 10:32:18 AM

PS C:\users\ramseyjw\Desktop>
```

Administrator: Windows PowerShell

```
PS C:\users\ramseyjw\Desktop> Get-Help Get-Date

NAME
    Get-Date

SYNOPSIS
    Gets the current date and time.


SYNTAX
    Get-Date [-Format <string>] [[-Date] <DateTime>] [-Day <in
     [-Minute <int>] [-Month <int>] [-Second <int>] [-Year <in

    Get-Date [-UFormat <string>] [[-Date] <DateTime>] [-Day <i
```

ETSU

Aliases

Like Linux, many Windows commands and cmdlets are mapped to aliases

Administrator: Windows PowerShell

```
PS C:\users\ramseyjw\Desktop> get-alias

CommandType     Name                                               Definition
-----------     ----                                               ----------
Alias           % ForEach-Object
Alias           ? Where-Object
Alias           ac Add-Content
Alias           asnp Add-PSSnapIn
Alias           cat Get-Content
Alias           cd Set-Location
Alias           chdir Set-Location
Alias           clc Clear-Content
Alias           clear Clear-Host
Alias           clhy Clear-History
Alias           cli Clear-Item
Alias           clp Clear-ItemProperty
Alias           cls Clear-Host
Alias           clv Clear-Variable
Alias           compare Compare-Object
Alias           copy Copy-Item
Alias           cp Copy-Item
Alias           cpi Copy-Item
Alias           cpp Copy-ItemProperty
Alias           cvpa Convert-Path
Alias           dbp Disable-PSBreakpoint
Alias           del Remove-Item
Alias           diff Compare-Object
```

Using PowerShell Cmdlets

Get-Command, Get-Member, Show-Command, New-Object

East Tennessee State University
Department of Computing
Jack Ramsey, Lecturer

Anatomy of a Cmdlet

Not compiled programs

Many Cmdlets preinstalled with OS

Cmdlets return objects, not strings

Can pipe output from one Cmdlet as input to another Cmdlet

Use get-help to explore or online documentation for examples

A simple example

`Get-Childitem` output is the same as the `dir` command

There is an alias that maps `dir` to `Get-Childitem` Cmdlet

```
Get-ChildItem c:\ | Format-List

Get-ChildItem c:\ | Format-List -Property name
```

Administrator: Windows PowerShell

```
PS C:\users\ramseyjw\Desktop> get-alias dir

CommandType     Name                                               Definition
-----------     ----                                               ----------
Alias           dir                                                Get-ChildItem

PS C:\users\ramseyjw\Desktop>
```

Formatting output of a Cmdlet

Format-List, Format-Wide, Format-Table

All Cmdlets can accept piped input from another Cmdlet

```
Get-ChildItem c:\windows | Format-Wide

Get-ChildItem c:\windows | Format-list

Get-ChildItem c:\windows | Format-table
```

Out-GridView

Creates a floating interactive table

Use the Get-Process Cmdlet

Get-help gps -> notice the name and aliases

```
gps | Out-GridView

gsv | ogv
```

`get-help ogv` -> what is ogv?

Get-Process modifying output

```
Get-Process

Get-Process | Get-Member

Get-Process | Sort-Object cpu

Get-Process | Sort-Object cpu -descending

Get-Process | Sort-Object cpu -descending | ogv

Get-Process -Name lsass
```

Get-Command Cmdlet

Use to access information about every command that is available

```
Get-Command *
```

```
gcm Get-Command | Format-List *
```

Returns all the properties for the get-command

```
(gcm Get-Command).Definition
```

Accessing the object's definition property

Administrator: Windows PowerShell

```
PS C:\users\ramseyjw\Desktop> (gcm Get-Command).Definition
Get-Command [[-ArgumentList] <Object[]>] [-Verb <String[]>] [-Noun <String[]>] [-Module <String[]>] [-TotalCount <Int32
>] [-Syntax] [-Verbose] [-Debug] [-ErrorAction <ActionPreference>] [-WarningAction <ActionPreference>] [-ErrorVariable
<String>] [-WarningVariable <String>] [-OutVariable <String>] [-OutBuffer <Int32>]
Get-Command [[-Name] <String[]>] [[-ArgumentList] <Object[]>] [-Module <String[]>] [-CommandType <CommandTypes>] [-Tota
lCount <Int32>] [-Syntax] [-Verbose] [-Debug] [-ErrorAction <ActionPreference>] [-WarningAction <ActionPreference>] [-E
rrorVariable <String>] [-WarningVariable <String>] [-OutVariable <String>] [-OutBuffer <Int32>]

PS C:\users\ramseyjw\Desktop>
```

`Get-Member` Cmdlet

Provides member information about particular objects that are returned from Cmdlets

Returns all methods and properties of objects

```
Get-ChildItem c:\ | Get-Member
```

```
gci | gm -MemberType Properties
```

Properties for the Get-ChildItem cmdlet

Verbs!

PowerShell uses a verb-noun naming convention

```
Get-Verb
(Get-Verb).Count
Get-Verb *un
```

Common patterns:

Add/Remove, Enter/Exit, Get/Set, Select/Skip

Verbs!

Administrator: Windows PowerShell

```
PS C:\users\ramseyjw\Desktop> Get-verb

Verb                                                                  Group
----                                                                  -----
Add                                                                   Common
Clear                                                                 Common
Close                                                                 Common
Copy                                                                  Common
Enter                                                                 Common
Exit                                                                  Common
Find                                                                  Common
Format                                                                Common
Get                                                                   Common
Hide                                                                  Common
Join                                                                  Common
Lock                                                                  Common
Move                                                                  Common
New                                                                   Common
Open                                                                  Common
Pop                                                                   Common
Push                                                                  Common
Redo                                                                  Common
Remove                                                                Common
```

PowerShell Scripting

More command line Kung-Fu

East Tennessee State University
Department of Computing
Jack Ramsey, Lecturer

Scripting in PowerShell

Commands typed or commands in a script are the same

To run scripts from the PowerShell window you need to config the execution policy

```
Set-ExecutionPolicy RemoteSigned
```

On Win 8, open a cmd prompt with admin rights, then run PowerShell

A Simple Script

Count of the handles for running processes

```
$handleCount = 0
foreach($process in Get-Process) { $handleCount +=
$process.Handles }
$handleCount
```

Place in a txt file with extension .PS1

Run at command line and as a script

Notice the loop control

Providers

PowerShell ***providers*** allow for interaction with data stores using familiar syntax

Example – Windows Registry

- Examine the \Windows\ hive

```
Set-Location HKCU:\Software\Microsoft\Windows\
Get-ChildItem
```

- Examine the autoruns key

```
Set-Location .\CurrentVersion\run
Get-ItemProperty .
```

Command Line Processing

`"Hello World"` simply returns Hello World

Math

Simple … (5+9)/2

Complex … `[System.Math]::`

`::` refers to a static method (system call)

```
#Num = 49
[System.Math]::Sqrt($Num)
```

```
(New-TimeSpan -Days 3).TotalHours
```

Command Line Processing

Administrator: Windows PowerShell

```
PS C:\users\ramseyjw\Desktop> "Hello World"
Hello World
PS C:\users\ramseyjw\Desktop> $Num = 49
PS C:\users\ramseyjw\Desktop> [System.Math]::Sqrt($Num)
7
PS C:\users\ramseyjw\Desktop> (new-TimeSpan -Days 3).TotalHours
72
PS C:\users\ramseyjw\Desktop>
```

More Math Functionality

```
[math] | Get-Member -Static -MemberType method
```

Listing of all Math Methods

```
[Math]::power(2,3)
```

```
0..7 | Foreach-Object {write-host $_
([math]::pow(2,$_)) }
```

Piping input to a loop

```
[math]::pi
$pi = [math]::pi
2*$pi
```

More Math Functionality

Administrator: Windows PowerShell

```
PS C:\users\ramseyjw\Desktop> 0..7 | ForEach-Object {write-host $_([math]::pow(2,$_))}
0 1
1 2
2 4
3 8
4 16
5 32
6 64
7 128
PS C:\users\ramseyjw\Desktop> $pi = [Math]::pi
PS C:\users\ramseyjw\Desktop> 2*$pi
6.28318530717959
PS C:\users\ramseyjw\Desktop>
```

Variables

Uses single equals for assignment **=**

Variable must start with **$**

Can assign strings, expressions, or commands

```
$var1 = 100
$var2 = 2
$var1/$var2
```

50

```
$array = "a"
$array = $array += "b"
```

ab

Arrays

PowerShell arrays are maintained by the system

- No explicit declaration
- Resized automatically

```
$array1 = 1,2,3,4

$array2 = “one”, “two”, “three”

$array3 = 1..100
```

Arrays (2)

```
$a

$a = 1, "A", (Get-Date)
```

Typed Arrays

```
[int[]] $a = 1,2,3,4
$a = $a + "test"
```

*** will error out ***

Arrays (3)

Combining arrays

```
$a1 = “red”,”yellow”

$a2 = “blue”, “green”

$a3 = $a1 + $a2
```

Processing Text Files

Use the Get-Content cmdlet

```
Get-Content input.txt
```

Place content into an array

```
$a = get-content input.txt
```

Each line of the text file is an element of the array

```
$a = (get-content input.txt)[0 .. 1]
```

-OR-

```
Get-content input.txt -totalcount 2
```

Processing Text Files (2)

Last element of an array

```
$a = (get-content input.txt)[-1 .. -3]
```

-1 is the last element, -2 2nd to last, etc.

Sort the array

```
(Get-Content C:\Scripts\Test.txt) | Sort-Object
```

Searching text files for a string

```
Select-String C:\Scripts\*.txt –pattern "Search String"
```

Control Logic

`7 -eq 7` … returns true

`-eq` Equal to

`-lt` less than

`-gt` greater than

`-ge` greater than or equal

`-le` less than or equal to

`-ne` not equal to

Control Logic (2)

`-not`	Not
`!`	Not
`-and`	And
`-or`	Or

Control Logic (3)

```
$x = 2 #creates a variable x and assigns 2 as the value
  if ($x -eq 5) {Write-Host "Hello my name is Bob"}
    elseif ($x -eq 4) {Write-Host "Hello, my name is Sue"}
    elseif ($x -eq 2) {Write-Host "Hello, my name is Troy"}
    elseif ($x -gt 1) {Write-Host "Hello, my name is Mary"}
  else {"I have no idea what my name is?"}
```

Control Logic (3)

Administrator: Windows PowerShell

```
>>
PS C:\users\ramseyjw\Desktop> $x
2
PS C:\users\ramseyjw\Desktop> if ($x -eq 5) {Write-host "Hello my name is Bob"} '
>>     elseif ($x -eq 4) {Write-Host "Hello, my name is Sue"}
>>     elseif ($x -eq 2) {Write-Host "Hello, my name is Troy"}
>>     elseif ($x -gt 1) {Write-Host "Hello, my name is Mary"}
>>   else {"I have no idea what my name is?"}
>>
Hello, my name is Troy
PS C:\users\ramseyjw\Desktop>
```

Control Logic (4)

Do While Loop Syntax

do while

do {code block}

while (conditional)

```
$i = 1
do {Write-Host $i; $i++}
while ($i -le 5)
```

Output 1,2,3,4,5

Control Logic (5)

`Foreach` Loop

For each statement do something

Iterate through an array

```
$ints = @(1, 2, 3, 4, 5)
foreach ($i in $ints)
{Write-Host $i}
```

Output 1,2,3,4,5

References

- http://technet.microsoft.com/en-us/scriptcenter/dd772285.aspx
- http://blogs.technet.com/b/heyscriptingguy/