POWERSHELL

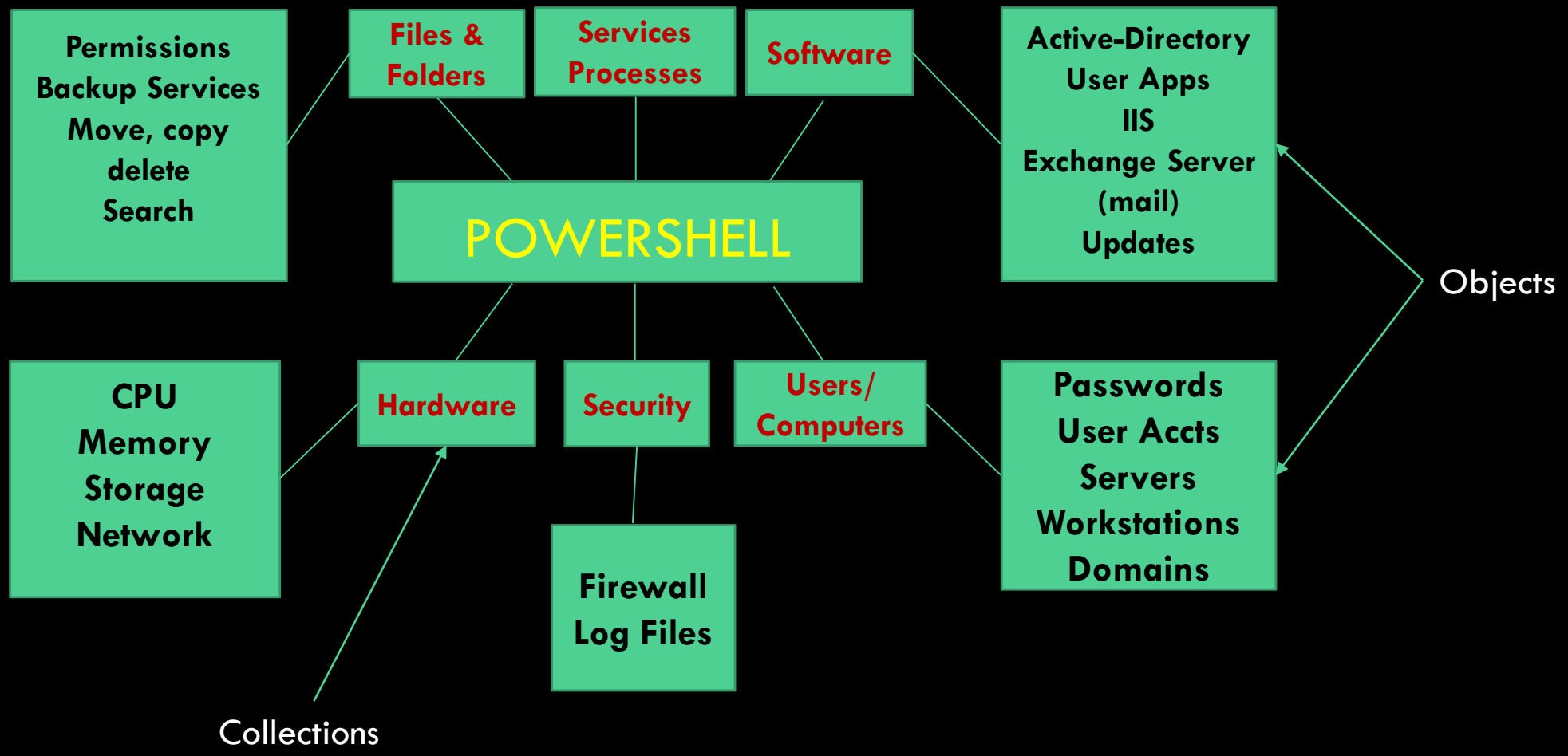
AN INTRODUCTION

WHAT IS POWERSHELL?

- An command-line, object-oriented, interactive scripting language designed to allow for the management and administration of both standalone systems as well as servers and workstations within a network environment.
- Provides an interface which allows for a greater degree of access and control to collections of objects and to individual objects, not provided within the typical “point and click” graphical environment.
- Administrators can access individual systems or multiple computers within a domain, from a remote location which greatly simplifies many administrative tasks required to maintain a healthy network environment.

Collections and Objects

PowerShell has access to all collections and objects within the Windows environment



WHAT IS POWERSHELL?

- PowerShell's scripting structure is based on "cmdlets"[2], which are themselves small scripts packaged into simplified commands. These "cmdlets" can be used on their own or incorporated into larger scripts. They provide administrators with a powerful set of tools to automate their daily activities.
- Typically scripts are relatively short programs designed to accomplish a single task. They range from a single line to hundreds of lines of code.
- Version 5.1 currently contains over 700 cmdlets and almost 900 functions. In addition, users can create their own cmdlets, import them from other users or download third party libraries (modules)[6].

HISTORY

- Following in the footsteps of MS-DOS, every version of Windows has incorporated a “command-line interpreter (cli)” as a means for users to interact with Windows underlying structure.
- In 2002, Microsoft began the development of a new command-line interface, namely PowerShell[3].
- 2006 saw version 1 of PowerShell released as an add-on. PowerShell 2 became the first permanently installed version in 2009.
- Version 2 and 3 were significant updates as the former added remoting[1] capabilities, while the latter introduced the ability to create Sessions.

HISTORY

- In 2016, Microsoft released the source code for PowerShell into the public domain (Open Source). It now incorporates support for OS X and Linux.
- Windows 10 systems are currently using PowerShell 5.1. Version 6 should be released as a Server Core version, which is a separate instance of PowerShell. It will take PowerShell to the “Cloud”.

WHO NEEDS IT?

- System administrators, power users or anyone the wants to automate procedures on their systems or network.
- Administrators that tend to Domain environments where access to a large number of remote systems is restricted by time, distance and cost.
- For large networks, Windows servers will typically be installed in “Server Core” mode. Network staff depend on PowerShell as there is no GUI available.
- The typical home user, honestly, will limited need for PowerShell. Still, having an understanding of the basic concepts of PowerShell provides users with a better understanding of how Windows works.

HOW IT IS USED

- Network administration staff use PowerShell to automate tasks such as:
  - Testing firewall status and software access through the firewall
  - Determining storage capacities and space available in all servers and user accounts
  - Administering Active Directory users accounts, passwords
  - Log file searches for problems within the network and other systems
  - Hardware/software inventory
  - System-wide software updates
  - Remote restart/shutdown of systems
  - Testing system security, searching for unauthorized software and files

CONNECTING TO USERS

- Prior to PowerShell, users applied commands and scripts using the Command prompt. It allowed users to interact with windows using Vbscript and older DOS commands. This shell will not run PowerShell cmdlets or scripts.
- While PowerShell is optimized for using cmdlets, it is also backward compatible with older DOS commands. It will also use aliases to help those familiar with Linux to use basic Linux commands[5] in PowerShell.
- Both 32-bit and 64-bit versions of PowerShell are installed in Windows 10. In addition, PowerShell provides users with the choice of using the typical command shell or the Integrated Scripting Environment (ISE) where longer scripts can be created and tested.

A WORD OF CAUTION

Before going on, there is one important warning:

***** PowerShell is very unforgiving *****

It does not care that you did not mean to delete all the files on your external drive and will not apologize for doing so. It does what you tell it to do and what it does is typically final.

Therefore, be very careful with cmdlets and understand what any cmdlet does before attempting to use it.

CMDLETS, PIPES AND OTHER THINGS

- Cmdlets[2] are the foundational structures in PowerShell. They simplify the manner in which users interact with Windows systems
- Cmdlets are structure in the verb-object format. For example, to list all files and folders in the current directory, the PowerShell command is **Get-childitem**. In addition, cmdlets typically have options (switches) that allow users to control how cmdlets interact with Windows.

  eg: **Get-childitem –recurse** # list all files and folder including sub-folders

- Cmdlets can be used alone or in combination with other cmdlets using pipes and longer scripts.

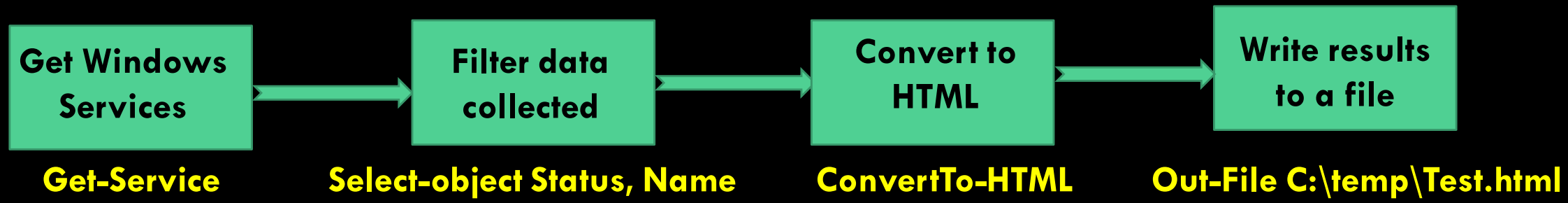
CMDLETS, PIPES AND OTHER THINGS

Pipes

- Imagine a well where water is sent through pipes for use in objects such as filters, washers and hot water heaters.
- PowerShell uses pipes to send data to data filters and objects such as cmdlets and functions when additional processing may be required. This processing can involve such things as filters, output formatting or even creating web pages.
- Without the ability to pipe data to other objects, PowerShell would lose much of its' usefulness.
- In PowerShell a Pipe is represented by the "|" character.

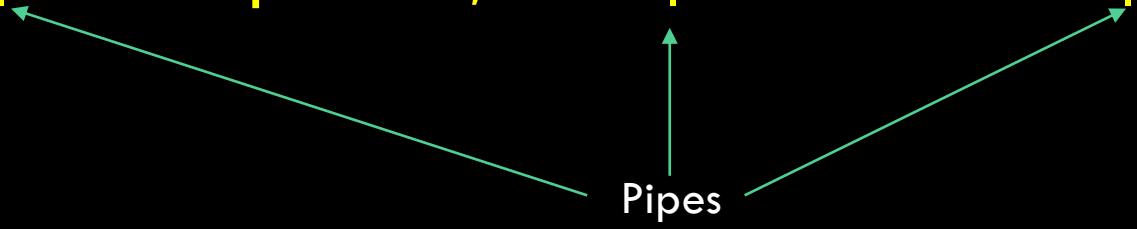
PIPES

In this example we want to get all of the services, filter the output, create a web page and write the HTML to disk.
Each section shows the PowerShell command (in yellow) required to complete this task. As each command is completed, the result is passed on to the next for processing



The completed script looks like this:[4]

```
Get-Service | Select-object Status, Name | ConvertTo-HTML | Out-File C:\temp\Test.html
```



CMDLETS, PIPES AND OTHER THINGS

Pipes

#This script will open two applications, wait for each to open then output a folder list

"calc","winword" | ForEach-Object {Start-Process $_} | Wait-Process ;dir

#This script will get all of the hard drives in the system, using Structured Query Language

GET-WMIOBJECT –query "SELECT * from win32_logicaldisk where drivetype = '3'"

#This script will get all of the files in the C:\temp folder and create a CSV file

Get-Childitem –file C:\temp | Export-CSV -Path C:\tmp\user.csv

Each pipe forwards data to the next part of the script. The number of cmdlets required depends on what the script is tasked to do.

CMDLETS, PIPES AND OTHER THINGS

Multiline Scripts

- In order to perform complex tasks, scripts typically require more code than the previous samples. These scripts may incorporate coding elements such as loops, conditional logic and functions to complete administrative jobs.
- Since scripts are usually designed to complete a single task and are therefore shorter than programs used to construct applications, Cmdlets help to reduce the size of scripts, which would contain many more lines of code without their use.

#Copy Ipod Music files and Change File Names Using Metadata

```
 ErrorActionPreference = "SilentlyContinue"
$ipodpath = gci  “e:\ipod_control\music\“ -hidden -file  -include *.m4a, *.mp3 -Recurse
$destination = “C:\tmp\"
$wmp = New-Object -ComObject wmplayer.ocx
gci -Path $ipodpath -hidden -file  -include *.m4a, *.mp3 -Recurse | foreach ($_) {
        $fileshortname = $_.Name.Split(".")[0]
        $filesuffix = $_.Name.Split(".")[1]
        $metadata = $wmp.newMedia($_)
        $ipodname = $fileshortname + "." + $filesuffix
        $finalname = $metadata.name + "." + $filesuffix
        copy-item $_ -destination $destination$finalname –whatif   }
```

LINKS AND REFERENCES

[1] Enabling Remoting

https://docs.microsoft.com/en-us/powershell/module/microsoft.powershell.core/enable-psremoting?view=powershell-5.1

[2] Useful Cmdlets

https://blogs.technet.microsoft.com/heyscriptingguy/2015/06/12/five-best-powershell-cmdlets/

[3] Command prompt vs Powershell

https://www.maketecheasier.com/difference-command-prompt-powershell/

[4] Create Web Page from Cmdlet Output

https://docs.microsoft.com/en-us/previous-versions/windows/it-pro/windows-powershell-1.0/ff730936(v=technet.10)

[5] PowerShell/Linux Commands

- http://cecs.wright.edu/~pmateti/Courses/233/Labs/Scripting/bashVsPowerShellTable.html

[6] PowerShell Modules

https://social.technet.microsoft.com/wiki/contents/articles/4308.popular-powershell-modules.aspx