Powershell

Introduction

Powershell Versions

- Windows Powershell version 5.1 is the target version for this course
- The get-host command can be used to see your Windows Powershell version
  (get-host).version.tostring()
- If you do not have version 5.1 of Windows Powershell, upgrade your version of Powershell
- Windows Powershell 5.1 is the current release of the Windows-specific version of Powershell
- Powershell Core 6 for Linux, MacOSX, and Windows was the first release of Powershell Core from Microsoft and has serious changes for Windows Powershell users
- Powershell Core 6 is so different they came up with a new command to run it (pwsh) and renamed the old Powershell to Windows Powershell - we will just use the name Powershell to save slide real estate and it will mean Windows Powershell for the duration of this course

Powershell Github Setup (optional)

- Clone your github COMP2101 repository to your PC and make a folder in the cloned folder to hold your Powershell scripts
- Create your scripts during the semester in that Powershell folder and keep it synchronized with github using git add, commit, push or the windows git tools from github

Powershell vs. Traditional Command Line Shells

- Traditional command line interface shells like bash, DOS cmd, etc. are tools that let you run commands found in the system and deal with text or simple numeric data
- Traditional shells only do what you tell them to do
- Traditional shells run commands in scripts the exact same way they run on the command line
- Traditional shells run scripts in their own processes

- Powershell is designed to run cmdlets and deal with objects
- Powershell guesses what you might have meant and does whatever it decides you wanted or thinks you should have wanted
- Powershell scripts may or may not run cmdlets differently in scripts from the command line and the command line behaves differently depending on how you start powershell
- Powershell runs scripts in a single process so data and output formatting bleeds between scripts run from the same command line

Powershell Privileged User

- Your Windows login provides a privilege level
- Windows administrator login does not grant Powershell administrator privilege
- Use Run As to get administrator privilege level in Powershell, regardless of what login you used for Windows
- BEWARE: Run As will only sort of make you Administrator if you are using active directory, and is silently dependent on active directory group policies

Privilege Exercise

- Start Powershell in console mode
- Run the command

```
get-acl c:/windows/*
```

- Note the error
- Run Powershell using Run As to gain administrator privilege and rerun the command
- Note the difference in the window frame title
- BEWARE: commands that change things can fail partway through and leave things in a broken state

Console vs. ISE

- Console mode is available even without the gui, and is especially useful when you have a low resolution display
- ISE (Integrated Scripting Environment) is a development environment and provides convenient access to supplemental tools
- Privilege restrictions apply to both
- They have separate profiles, most commands work in both
- Only console mode has a future and is cross-platform as of Powershell Core 6; Microsoft's family of IDE products are to be used with PowerShell going forward (VSCodium is a good way to get started)
- ISE is deprecated now, and only works with the old Windows Powershell 5 and below

Interface Exercises

- Start Powershell in console mode and in ISE mode
- Run the command ise from the Powershell console
- Try entering these commands in both modes and look for differences in the output

```
get-process -id $pid

get-host

get-history
```

Cmdlets

- Cmdlets are what the light-weight commands in Powershell are called, Powershell does not start new processes to run them
- Thousands are built into Powershell and you can create your own by writing functions
- Cmdlets and their parameters are case insensitive

Cmdlet Names

- Which Powershell cmdlets are available depends on the .NET libraries and are therefore dependent on the .NET version you have installed, as well as what operating system you have
- The general form for cmdlets is verb-noun
- The well-known verbs can be displayed with get-verb
- Nouns are defined by Powershell and the installed modules from .NET along with any modules you have installed
- Available commands can be displayed by the get-command cmdlet

Getting Help

- Powershell provides online help with the get-help command
- help is a function invoking get-help that automatically paginates the output by piping get-help to the more command
- Running help or get-help without any parameters displays how to use the get-help command

Help Types

- You can run get-help on cmdlets or on topics
- Topic help pages are named about_topic, cmdlet help pages are named cmdlet

  e.g. help about_

  e.g. help get-date

Default Help

- By default, help only displays basic help including DESCRIPTION, SYNTAX, and SEE ALSO sections
- Like most cmdlets, get-help accepts several parameters which modify how it works and what it displays
- Powershell only includes the basic help in the default installation
- More help content is available for most cmdlets by using the -Detailed, -Examples, and -Full parameters
- BEWARE: these options don't work properly unless you run update-help at least once on the computer

Updating Help

- Use the update-help cmdlet to install complete help pages and keep them up to date
- update-help will only update your pages once a day unless you use the -Force parameter
- update-help requires administrator privilege
- update-help should be added to scheduled tasks if you keep local help pages
- BEWARE: update-help should be run with erroraction set to deal with the fact that Microsoft doesn't keep the updated help servers complete

Online Help

- The -Online parameter can be used to view the latest help for cmdlets and topics on the web, without running update-help on your own computer
- e.g. help -Online get-help
- The online help includes the ability to choose which Powershell version to look at for help because cmdlets can change from one version to another
- Powershell online help does not provide Powershell 1.0 or 2.0 help
- BEWARE: the online help does not automatically choose the current version of Powershell to show help for

Help Exercises

- Use get-help about_ to view the available topics list
- Try viewing the topic help for command syntax, pipelines, and parameters
- Use get-help with the start, stop, clear, get, and set verbs only to see what nouns are available for those verbs
- Use get-help to get some descriptions for the following sample cmdlets:
- get-process, get-date, get-host, clear-host, stop-job, start-service

Extending Help Exercises

- Run the update-help cmdlet to install full help pages on your computer
- Compare the output for the help get-date cmdlet when using the help cmdlet with no parameters versus using the -detailed, -examples, and -full parameters
- Compare the help -full get-date output to the online version from help -online get-date
- Use show-command to try the help popup and compare it to the command help pane in ISE

ISE Help

- The show-command cmdlet will display a popup window which allows click-based command construction
- You can access help from the show-command popup
- The show-command popup captures input
- The show-command popup is implemented as a pane in ISE, which does not capture the input

Tab Completion

- Parameters in Powershell are words starting with a - character
- Both commands and their parameters can be completed using the tab key
- Repeatedly pressing tab cycles alphabetically through matching choices
- Shift-tab moves backwards through the list of choices
- The list wraps around at both ends
- Pressing Control-space shows a list of possible completions

Parameters in Scripts

- Parameters only require you to type enough characters to uniquely identify a specific parameter
- Cmdlets which require parameters to run will complain when you try to run them without the required parameters
- Parameters can be organized in named sets to avoid conflicts between mutually exclusive parameters
- Always use complete parameter names in scripts
- See about_Parameters for more info

Command Completion Exercises

- In a Powershell console window, try using tab to view all possible parameters for the get-date cmdlet
- In ISE, observe what happens as you type commands and their parameters, use get-random as your sample command for this
- In ISE, use the command list pane to create and run a get-date command that displays the date with day set to 1, hour set to 2, minute set to 3, month set to 4, and year set to 5
- Use control-left click on the cmdlet name in the command list pane to dismiss the cmdlet entry subpane

Execution of Scripts

- On Windows, execution policy determines whether scripts can be run as commands
- Execution policy has scope, there are separate process, user, and system scopes available
- The file extension is used to determine if a file contains a Powershell script
- The extension ps1 means a Powershell script
- BEWARE: Powershell runs scripts in the current process meaning the commands you run and scripts you execute affect each other in unexpected ways

Execution Policy

- Execution Policy is retrieved using get-executionpolicy
- Execution Policy is set using set-executionpolicy policy (using Run As Administrator) where policy is one of several choices: restricted, allsigned, remotesigned, unrestricted, bypass
- The default policy is restricted, up to 5.1 and prior to Windows Server 2012R2, remotesigned after that
- See about_Execution_Policies for more info
- Execution Policy only exists in Windows
- BEWARE: remotesigned is only meaningful if every machine which has stored or runs the script is a windows machine with an NTFS filesystem

Execution Policy Exercises

- Use get-executionpolicy to see what your policy is currently set to
- Try the -list parameter to see what it is set to for different scopes
- Create a file named helloworld.ps1 with one line it that looks like this:

```
“Hello World!”
```

- Try to run your helloworld.ps1 script as a command
- Use set-executionpolicy to set your policy to a mode that allows you to run scripts
- Rerun your script as a command

Command Path

- Like bash, Powershell has a path variable that defines where the shell looks for commands using a semicolon-delimited list of folder names called $env:PATH
- Powershell provides a default command path stored in the variable
- To see what is in the variable, type the variable name on the command line
- To change it, type $env:PATH = "$env:PATH;drive:/new/path/name/to/add"
- You can create a profile file to run startup commands, which is how you might choose to set a different path that takes effect every time you run powershell

Profiles

- Powershell has several recognized profile files
- To see the name of the profile file that applies to your current session, look in the $profile variable
- To see if you have a profile file, run test-path $profile
- To create such a file, try notepad $profile
- You can also create a profile file using

```
new-item -itemtype file -force $profile
```

- See about_Profiles for more info

Profile Exercises

- Clone your github repository if you haven't already done that, and move your helloworld.ps1 script to a directory in your cloned repository
- Add a line to your $profile file on your PC that adds your cloned repository's powershell scripts directory to your $env:path
- Start a new powershell and verify you can run helloworld.ps1 without entering a path to the command