# Powershell

Working with Objects
COMP2101 Fall 2018

# Objects

- An object is a data structure residing in memory
- That structure has places for code and data, those things are called members of the object
- Object references can refer to one object or to a collection of objects
- Objects can do things concurrently and independently

# Object Members

- Code we access in an object is called a method
- Data we access in an object is called a property
- Properties can be objects or collections of objects
- Data in Powershell has a type which guides us in handling that data

# Working With Objects

- Objects get created by cmdlets
- The cmdlet that creates an object has a handle (reference) for it
- That cmdlet can decide to drop the handle, or can pass it to the shell as output, which by default powershell will format and display as text
- Objects continue to exist as long as anything has a handle for them
- Using cmdlets to create objects and use them as command line data for other cmdlets can be done by putting () around the creation cmdlet

```
1..254 |
ForEach -Process {WmiObject -Class Win32_PingStatus -Filter ("Address='192.168.16." + $_ + "'") -ComputerName .} |
Select-Object -Property Address, StatusCode | ft -auto
```

# Objects On The Command Line

- Objects produced by cmdlets are normally displayed by the shell, some cmdlets produce objects you might not expect (e.g. mkdir)
- Object contents can be sent places using redirection
- >, >> are similar to bash output redirection, they discard the object handle(s) after writing the object(s) content to the file
- The | symbol creates a pipeline to transfer object handles from one cmdlet to another
- The out-*placename* cmdlet can be used to send object contents to other places, discarding the handles after sending - *placename* can be nouns like *null* or *file*
- Object handles can be assigned to variables

# Objects Examples

- Create a string object, let the shell display it
  "my string object"

- Create a datetime object, let the shell display it
  get-date
  (get-date)

- Create a collection of objects, let the shell display it, then try saving them
  "my object1","my object2",(get-date),362,(get-host),"my object3"
  "turkey","chicken","mouse","string" > food

- Use > to send the content of a datetime object to a file
  get-date > c:/mydate.txt
  and then examine the file

- Use new-item to make a directory
  new-item -itemtype directory c:/mytmp

- Use ">" to save the output of the new-item cmdlet above
  new-item -itemtype directory c:/mytmp2 >c:/mytmp2.txt
  and see what gets saved to the file

- Use the out-null cmdlet to discard the object handle produced by the new-item cmdlet
  new-item -itemtype directory c:/mytmp3 | out-null

# Out Verb Cmdlets

- Several useful destinations are available for objects
- out-host can be used to place output objects into the same output stream as write-host
- out-null throws away any objects passed to it
- out-file saves objects to files in a more sophiscated way than the simple > redirect
- out-gridview display objects in a spreadsheet-style popup window
- out-printer sends object to a printer

# Out Verb Cmdlets Exercises

- get-process | format-table * -autosize > procs.txt
- get-process |
  format-table * -autosize |
  out-file -width 300 wideprocs.txt
- get-process | out-null
- get-process | out-gridview
- get-process | select * | out-gridview

# Aliases and Functions

- Like bash, Powershell supports aliases and functions
- Functions in Powershell can be simple like in bash, or can be written to behave the same as full cmdlets
- Many aliases and functions are predefined by Powershell for you
- Many basic UNIX commands are aliased or implemented in functions to make life easier

# Aliases Examples

- Use get-alias to see the list of predefined aliases, how many UNIX commands can you spot in the list?
- Note that not all cmdlets produce the same output or work the way you might expect for the aliased UNIX commands, e.g. ls, rm, mkdir
- Aliases support command line parameters, try
  mkdir c:/mytmp4
- Create an alias for notepad.exe called np
  new-item -path alias:np -value notepad
- Remember when scripting to discard objects the user wouldn't expect to see

# Functions Examples

- You can list the defined functions
  ls function:

- You can view the content (code) of a function
  gc function:more

- You can create trivial functions
  function myfunc { "this is my function" }
  myfunc
  ls function:
  gc function:myfunc

# Pipelines

- A pipeline can accept object handles from a cmdlet and pass them to another cmdlet
- A command line can have multiple pipes
- Cmdlets in a pipeline can choose whether or not to use objects passed to them by a pipe, to pass them along in the pipe, or to drop their handles
- Any objects produced by the last cmdlet in a pipeline get displayed by the shell

# Pipeline Examples

- "c:/windows" | ls
- get-process | more
- get-process | sort cpu | more
- mkdir c:/mytmp5 | out-null

# Get-Member

- The get-member cmdlet displays a list of the stuff in an object that we can retrieve, store, or invoke, just pipe an object to it
- Every object has a type, get-member shows us the type of the object
- We can retrieve properties or set new data into them, similar in concept to variables
- We can invoke methods in objects to cause objects to perform some task for us
- Besides properties and methods, objects can actually have lots of other kinds of stuff in them, such as aliases, noteproperties, scriptproperties, etc.

# Object Members Examples

- Use get-member to view the members of objects
  get-host | get-member
  get-member -inputobject (get-date)
  get-date | get-member -membertype properties
  get-date | get-member -membertype property
  get-process | get-member | more
  get-wmiobject -class win32_process | get-member | more

- Note that each property in an object has a data type

- Note that each method in an object has a data type and may accept parameters, each of which has a type

- -MemberType parameter can be used to retrieve only specific types of members

# Methods

- Methods are named blocks of code contained in objects
- Methods can be passed data as parameters
- Methods can return typed data
- Methods can be invoked using dot notation
- When a method is invoked, the object itself performs the task by running its code, not Powershell

# Dot Notation

- Members of an object can be accessed using the object handle, then a dot, then the member name
- You can get an object handle using the (cmdlet) syntax
- (get-date) gives you the handle of the object produced by the get-date command
- (get-date).millisecond retrieves the property millisecond from the object produced by get-date
- (get-date).adddays(5) invokes the adddays method to add 5 days to the datetime object produced by get-date

# Dot Notation Exercises

- (get-date).gettype()
- ("test").gettype()
- (5).gettype()
- ("a","b","c").length
  ("a","b","c").count
- (get-date).dayofweek
  (get-date).dayofweek | get-member
- get-process powershell
  get-process powershell | format-list *
  (get-process powershell).startinfo
  (get-process powershell).startinfo.environmentvariables
- (gwmi -class win32_process).getowner()
  (gwmi -class win32_process).getowner().user

# Creating Custom Objects

- Objects can be created on the command line by specifying data and letting powershell decide what to create
- Objects can be created by cmdlets
- A useful cmdlet for making objects of your own design is
  new-object -typename psobject -property @{name=value;name2=value2}
- Multiple names and values can be specified, and placing them on separate lines makes them easy to read
- This can be helpful for creating objects that have a custom set of members, particularly if you are building objects in a loop
- Predefined objects can be created by specifying a typename for those objects

# Custom Objects Examples

- new-object -typename psobject -property @{key1="value1";key2="value2";key3=(get-date).millisecond}

- (get-date).dayofweek | get-member -membertype property
  new-object -typename system.dayofweek -property @{value__=3}

- foreach ($c in (1,2,3,4,5,6,7,8)) {
  new-object -typename psobject -property @{
  PlaceCount=$c;
  MaxValueInBinary=[math]::pow(2,$c);
  MaxValueInOctal=[math]::pow(8,$c);
  MaxValueInHex=[math]::pow(16,$c)
  }
  }

# Format-Table

- Table is the default format for many cmdlets that display collections of objects (e.g. get-process, get-alias, get-eventlog), but not all
- format-table can be used to display non-default properties or format them to suit your requirements, you can specify property names to be displayed
- ft is an alias for format-table
- -AutoSize parameter very helpful
- format-table is designed to only be used in pipelines at the end

# Format-Table Examples

- get-date | format-table
- get-date | format-table -autosize
- (get-date),
  (get-date).adddays(4),
  (get-date).addhours(16) |
  format-table -autosize year, month, day, hour, minute
- get-date | format-table | get-member

# Format-List

- List is the default format for many cmdlets that display single objects (e.g. get-host, get-member, get-service), but not all
- format-list can be used to display different data items, you can specify property names to be displayed
- fl is an alias for format-list
- fl * is a way to see the data for all printable properties on an object

# format-list Exercises

- get-process powershell
- get-process powershell | format-list
- get-process powershell | format-list *
- get-process svchost
- get-process svchost | format-list id, name
- get-wmiobject -class win32_process | format-list processid, name, commandline