Working with Objects
Summer 2022

Powershell

Familiar Commands, Unfamiliar Data

- Many commands from Linux appear to be available in Powershell
- They are actually aliases or functions which have Linux names and actually run Powershell commands
- So these commands look familiar, but work differently from the commands they are supposed to look like
- Knowing what objects are and how to work with them is required to use even these commands
-

clear	
man	history
echo	kill
cd	diff
ls	lp
cat	mount
more	ps
head	pwd
tail	sleep
cp	sort
mv	tee
rm	curl
mkdir	wget
rmdir	

Basic File Handling

- Working with files to do simple things can feel similar to Linux bash, but it is not
- cd is used to change your directory but unlike bash, which takes you to your home directory when you just enter cd without a target, powershell does not change directory unless you give it a target, ~ or $home can be used to specify your home directory
- ls will list files with a similar output to the old DOS dir command, but behaves differently and produces different output from the actual ls command, and requires using object collections and a different way of thinking to work with anything but a simple list of a single directory
- In general, it is best to use the Powershell cmdlets instead of these Linux-like commands to avoid confusion and unexpected results
- Either way, the output from Powershell commands is always zero or more objects, never just plain text, and you may not have any use for the objects they produce

set-location
get-childitem
new-item
remove-item
move-item
copy-item
get-content
set-content
add-content

Objects

- An object is a data structure residing in memory
- That structure has places for code and data and other things
- The code, data, and other things in an object are called members of the object
- Objects can be one object or a collection (a.k.a. array) of objects

Object Members

- Code we access in an object is called a method
- Data we access in an object is called a property
- Properties are objects or collections of objects
- Data in Powershell has a type which guides us in handling that data
- Powershell objects may have a default output format or may have multiple default output formats or may have no default output format
- BEWARE: Powershell object display is wonky and unpredictable, so always explicitly control your output in scripts and always make your scripts only produce one type of output object whenever possible

Working With Objects

- Objects get created by cmdlets
- That cmdlet can decide to release the object, or can pass it to the shell as output, which by default powershell will format and display as text using unpredictable rules
- Objects continue to exist as long as anything refers to them, objects we want to keep are usually assigned to variables, which puts the object reference (sometimes called a handle) into the variable
- We can get the handle for any object by putting () around the object or a pipeline that creates one or more objects

Objects On The Command Line

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

Objects Examples

- Create a string object, let the shell display it
  "my string object"
- Create a collection of objects, let the shell display it, then try saving a similar collection to a file with output redirection
  "my object1","my object2","my object3"
  "turkey","chicken","mouse","string" > $home/desktop/food
- Use > to send the display output of a get-date cmdlet to a file, then examine the file
  get-date > $home/desktop/mydate.txt
- Use mkdir to make to make directories in different ways
  mkdir a
  mkdir a b c
  mkdir a,b,c
  mkdir "a","b","c"
  mkdir ("a","b","c")
- Use the out-null cmdlet to discard the object produced by mkdir
  mkdir d | out-null

Out Verb Cmdlets

- By default, when a command produces objects and does not specify a destination for them, Powershell displays them on the screen
- Objects can be sent to other destinations
- out-null discards objects
- out-file saves objects to files in a more sophisticated way than the > redirect
- out-gridview display objects in a spreadsheet-style popup window
- out-printer sends objects to a print queue

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 command names are aliased or implemented in functions
- Beware when using variables in functions, existing local variables get copied to function variables

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 $home/desktop/mynewtmpdir
- 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

- Pipe an object to the get-member cmdlet to display a list of the members in an object that we can retrieve, store, or invoke
- Every object has a type, get-member shows us the type of the object
- Properties can be retrieved and sometimes changed, 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.
- Some cmdlets make objects that have extra hidden properties called adapted or extended properties which require you to know they exist and to handle them specially, get-member -view all can sometimes help you to at least know they exist

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.username
- (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 ($c in (1..4)) {
    new-object -typename psobject -property @{
        PlaceCount=$c;
        MaxValueInBinary=[math]::pow(2,$c);
        MaxValueInOctal=[math]::pow(8,$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

Lab 2 - creating and exploring objects