Practical 5: Introduction to PowerShell and querying processes on a machine

**Introduction**

In this lab we introduce PowerShell and also concentrate a little on processes. PowerShell is a relatively new command shell from windows. Windows PowerShell is Microsoft's task automation and configuration management framework, consisting of a command-line shell and associated scripting language built on .NET Framework. PowerShell provides full access to COM and WMI, enabling administrators to perform administrative tasks on both local and remote Windows systems.

**1. What can PowerShell execute**

What can windows PowerShell execute:

* cmdlets, which are .NET programs designed to interact with PowerShell
* PowerShell scripts (files suffixed by .ps1)
* PowerShell functions
* standalone executable programs

**2. Sampling some PowerShell commands**

See at <http://en.wikipedia.org/wiki/Windows_PowerShell>

**3. Calling .NET classes from PowerShell**

As already mentioned, PowerShell is a scripting language built on top of the .NET framework. Hence we can call .NET classes from PowerShell. Lets demonstrate by creating a .NET messagebox. To do this perform the following two steps:

# Load the assembly

[System.Reflection.Assembly]::LoadWithPartialName("System.Windows.Forms")

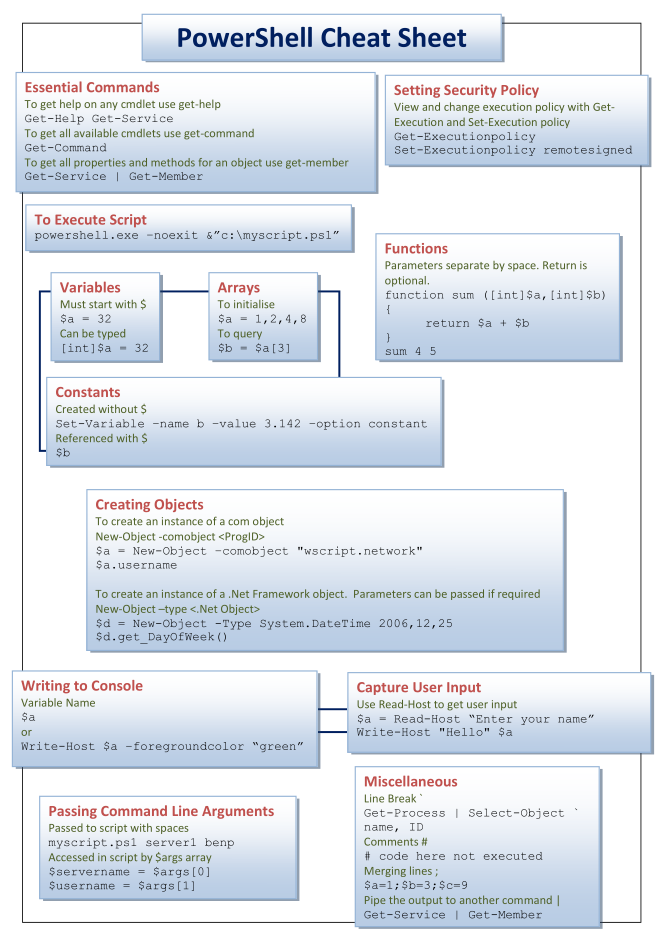
# Display the messagebox

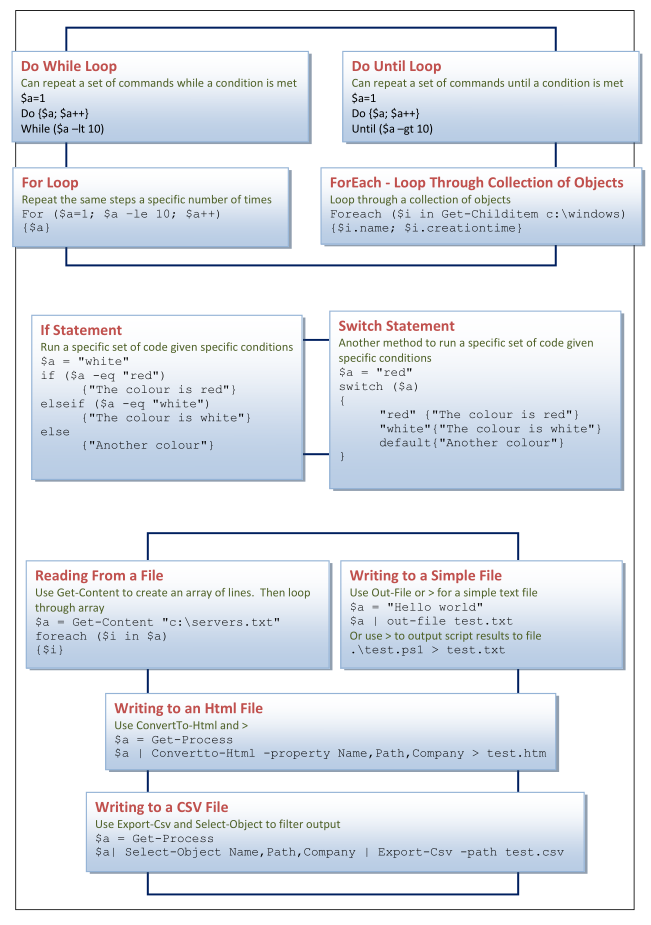
[System.Windows.Forms.MessageBox]::Show("Hello post grad class.")

**4. PowerShell ISE**

Simply demonstrate its use.

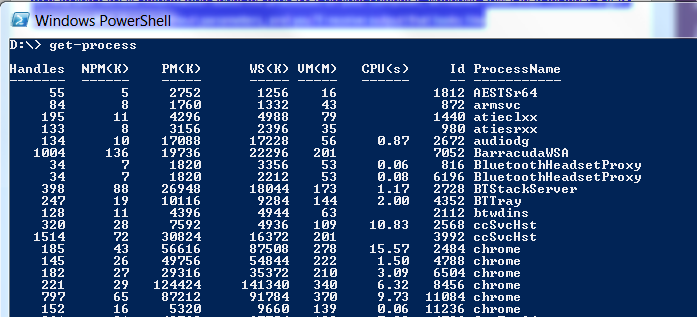
**5. Go through the PowerShell cheat sheet:**



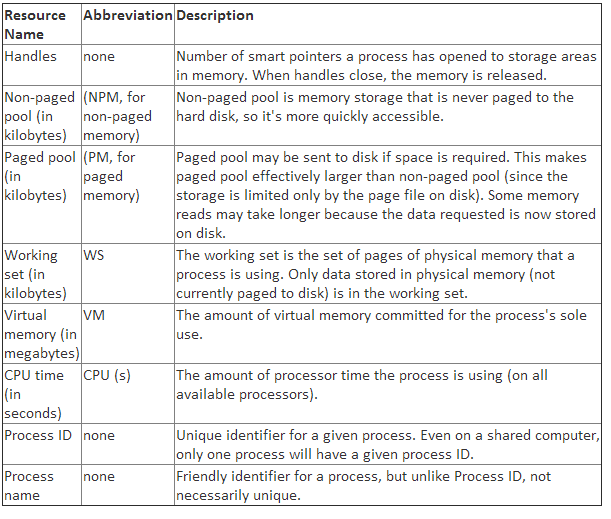


**6. Using PowerShell to get process information**

To help you retrieve information about the processes on your computer, PowerShell includes a **get-process cmdlet.** Run it without parameters, and you'll receive output that looks like



The abbreviations stand for the following:



**7. Using PowerShell to retrieve highest load processes**

You don't necessarily need a list of all the processes, but you *will* need a general idea of which processes are consuming the most resources. For example, a working set (see box above) of a processes serves as a good indicator of its memory stress on the system. To find all processes with a working set greater than 10MB, type the following command:

**get-process | where-object {$\_.WorkingSet -gt 10000000}**

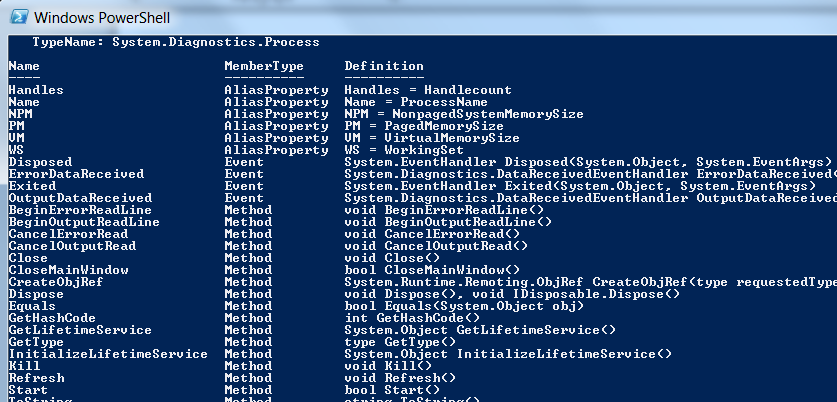
Note the following:

* Piping of commands
* Use of where-object
* Use of $\_ alias
* Use of -gt

**8. Using PowerShell to retrieve all process member information**

Note: There's quite a few more process properties available than meets the eye. To see them all and get their property names type:

**get-process | get-member**



Note the following:

* Methods and properties
* Aliases map to real properties
* Get-member shows what the process object has overall, but does not show actual values of properties

**9. Using PowerShell to query actual process values**

Firstly, get a particular process that is running:

**get-process –Id 1234**

To query actual process 1234 values:

**get-process -Id 1234 | select PeakWorkingSet**

**10. Using PowerShell to kill unnecessary processes**

Firstly, get a particular process that is firstly getting the processes you want to kill and then pipe this to the kill command:

**get-process –Id 1234 | kill**

Note that kill is an alias for stop-process. Also you can kill multiple processes at the one time:

**get-process w\* | kill**

**11. For submission**

Submit a written report on this lab to [billy.stack@staff.ittralee.ie](mailto:billy.stack@staff.ittralee.ie) that has:

* A description of what we covered in the lab
* A sample PowerShell script that performs something meaningful that includes
  + a control statement e.g. “if”, “switch” or “for”
  + one example of piping commands together
  + must be readily executable