PowerShell Cheat Sheet

PowerShell Cheat Sheet v2
Adapted with permission from Ben Pearce's Original.
http://sharepointjack.com/2013/powershell-cheat-sheet-v2-00/

Essential Commands

To get help on any cmdlet use get-help
```
Get-Help Get-Service
```
To get all available cmdlets use get-command
```
Get-Command
```
To get all properties and methods for an object use get-member
```
Get-Service | Get-Member
```

Setting Security Policy

View and change execution policy with Get-Execution and Set-Execution policy
```
Get-Executionpolicy
Set-Executionpolicy remotesigned
```

To Execute Script

```
powershell.exe -noexit &"c:\myscript.ps1"
```

Functions

Parameters separate by space. Return is optional.
```
function sum ([int]$a,[int]$b)
{
      return $a + $b
}
sum 4 5
```

Variables

Must start with $
```
$a = 32
```
Can be typed
```
[int]$a = 32
```

Arrays

To initialise
```
$a = 1,2,4,8
```
To query
```
$b = $a[3]
```

True and False

Set a Variable to true
```
$a = $true
```
Check if a Variable is false
```
If ($b -eq $false)
```

Constants

Created without $
```
Set-Variable -name b -value 3.142 -option constant
```
Referenced with $
```
$b
```

Creating Objects

To create an instance of a com object
New-Object -comobject <ProgID>
```
$a = New-Object -comobject "wscript.network"
$a.username
```

To create an instance of a .Net Framework object. Parameters can be passed if required
New-Object –type <.Net Object>
```
$d = New-Object -Type System.DateTime 2006,12,25
$d.get_DayOfWeek()
```

Writing to Console

Variable Name
```
$a
```
or
```
Write-Host $a -foregroundcolor "green"
```

Capture User Input

Use Read-Host to get user input
```
$a = Read-Host "Enter your name"
Write-Host "Hello" $a
```

Passing Command Line Arguments

Passed to script with spaces
```
myscript.ps1 server1 benp
```
Accessed in script by $args array
```
$servername = $args[0]
$username = $args[1]
```

Miscellaneous

Line Break `
```
Get-Process | Select-Object `
name, ID
```
Comments #
```
# code here not executed
```
Merging lines ;
```
$a=1;$b=3;$c=9
```
Pipe the output to another command |
```
Get-Service | Get-Member
```

Do While Loop

Can repeat a set of commands while a condition is met

```
$a=1
Do {$a; $a++}
While ($a –lt 10)
```

Do Until Loop

Can repeat a set of commands until a condition is met

```
$a=1
Do {$a; $a++}
Until ($a –gt 10)
```

For Loop

Repeat the same steps a specific number of times

```
For ($a=1; $a -le 10; $a++)
{$a}
```

ForEach - Loop Through Collection of Objects

Loop through a collection of objects

```
Foreach ($i in Get-Childitem c:\windows)
{$i.name; $i.creationtime}
```

If Statement

Run a specific set of code given specific conditions

```
$a = "white"
if ($a -eq "red")
      {"The colour is red"}
elseif ($a -eq "white")
      {"The colour is white"}
else
      {"Another colour"}
```

Switch Statement

Another method to run a specific set of code given specific conditions

```
$a = "red"
switch ($a)
{
      "red" {"The colour is red"}
      "white"{"The colour is white"}
      default{"Another colour"}
}
```

Reading From a File

Use Get-Content to create an array of lines. Then loop through array

```
$a = Get-Content "c:\servers.txt"
foreach ($i in $a)
{$i}
```

Writing to a Simple File

Use Out-File or > for a simple text file

```
$a = "Hello world"
$a | out-file test.txt
```

Or use > to output script results to file

```
.\test.ps1 > test.txt
```

Writing to an Html File

Use ConvertTo-Html and >

```
$a = Get-Process
$a | Convertto-Html -property Name,Path,Company > test.htm
```

Writing to a CSV File

Use Export-Csv and Select-Object to filter output

```
$a = Get-Process
$a| Select-Object Name,Path,Company | Export-Csv -path test.csv
```

Load a Snap In

Load a Snap in for added functionality, supressing error info if the snap in is already loaded.

$Add-PSSnapin microsoft.sharepoint.powershell -ErrorAction SilentlyContinue

Working With Shortened commands (Aliases)

Use Get-Alias to list out all commands with shortened alternatives

```
Get-Alias
```

Find the long form of a command from its alias:

```
Get-Alias –name dir
```

Find all aliases of a form of a command from its alias:

```
Get-Alias –Definition “Get-ChildItem”
```

Refining output

Where-Object (Where)

Where is used to limit the output of a command

Command | Where {$_.ParameterName –like “value”}

```
$a = dir |Where {$_.PSIsContainer –eq $true}
```

Sort-Object (Sort)

Limit which fields are returned

Long Form:

```
Dir | Sort-Object Name
```

Short Form:

```
Dir | Sort Name, Length
```

Select-Object (Select)

Limit which fields are returned

```
Dir | Select Name, Length
```

Limit how many results are returned

```
Dir | Select –First 3
```

Listing Details

Sometimes there is more than is shown by default

Format-List outputs more fields, in a list format

```
Dir | Format-list
Dir | fl
```

Chaining Multiple Commands

Multiple commands and refiners can be used to get just the right output:

```
Dir | where {$_.PSIsContainer –eq $true} | Sort name | Select –first 3
```

Learning about a result by using where, the dot (.) and tab

Some commands return complex results that can be further broken down.

It is often helpful to narrow down the results to just one item, and assign that one result to a variable so that you can inspect its properties.

```
$d = Dir #returns too much
$d = Dir | select –first #better, returns one entry
```

At this point you can type $d. and hit the tab key repeatedly to see the different properties.

```
$d.(tab) #starts to list the properties such as $d.name, $d.fullname
```

Another example, using **where** to pick the specific result to inspect

```
$d = Dir | Where {$_.name –eq “Windows”}
```