

# Windows PowerShell Quick Reference

## How to Access Arguments

To access command-line arguments used when starting a script use the automatic variable **\$args**. You can cycle through the individual arguments in the \$args collection by using code similar to this:

```
foreach ($i in $args) {$i}
```

To access a particular argument use the collection index number, with 0 representing the first item in the collection, 1 representing the second item, etc:

```
$args[0]
```

You can reference the *last* item in a collection by using the index number `-1`:

```
$args[-1]
```

## How to Use Colored Text

To display text in a different color use the **Write-Host** cmdlet and specify a foreground color:

```
Write-Host "test" -foregroundcolor "green"
```

You can also specify a different background color:

```
Write-Host "test" -backgroundcolor "red"
```

## How to Insert a Paragraph Return

To insert a paragraph return in your output use the newline character ``n`:

```
Write-Host "Line 1.`nLine 2."
```

## How to Write in Reverse Video

To echo a message in reverse video use the **Write-Warning** cmdlet:

```
Write-Warning "An error has occurred."
```

## How to Insert Comments

To insert a comment, use the pound sign (`#`):

```
# This is a comment, not a line to be run.
```

## How to Solicit Input

To solicit input from a user, use the **Read-Host** cmdlet, followed by the prompt to be displayed:

```
$a = Read-Host "Please enter your name"
```

## How to Insert Line Breaks

To insert a line break into a Windows PowerShell script use the backtick (```):

```
Write-Host `
    "This is a continuation of the line."
```

You can also break a line at the pipe separator (`|`) character (assuming your line uses the pipeline):

```
Get-ChildItem C:\Scripts |
    Sort-Object Length -Descending
```

## How to Create Multi-Command Lines

To put multiple commands on a single line, separate those commands using a semicolon:

```
$a = 1,2,3,4,5; $b = $a[2]; Write-Host $b
```

## How to Make Comparisons

Windows PowerShell cmdlets (like **Where-Object**) use a special set of comparison operators, including those shown in the following table.

Each of these operators can be made case sensitive by adding a **c** immediately after the hyphen. For example, **-ceq** represents the case-sensitive equals operator; **-clt** is the case-sensitive less than operator.

<b>-lt</b>	Less than
<b>-le</b>	Less than or equal to
<b>-gt</b>	Greater than
<b>-ge</b>	Greater than or equal to
<b>-eq</b>	Equal to
<b>-ne</b>	Not equal to
<b>-like</b>	Like (uses wildcards for matching)
<b>-notlike</b>	Not like (uses wildcards for matching)

# Windows PowerShell Quick Reference

## How to Read a Text File

To read the contents of a text file into a variable, call the **Get-Content** cmdlet followed by the path to the text file:

```
$a = Get-Content C:\Scripts\Test.txt
```

Each line in the file ends up as an item in the array **\$a**. If you want to access a single line in the file you can simply specify the index number corresponding to that line:

```
$a[0]
```

This command echoes back the *last* line in **\$a**:

```
$a[-1]
```

**Bonus.** To determine the number of lines, words, and characters in a text file use this command:

```
get-content c:\scripts\test.txt |
measure-object -line -word -character
```

## How to Write to a Text File

To save data to a text file use the **Out-File** cmdlet:

```
Get-Process | Out-File C:\Scripts\Test.txt
```

To append data to an existing file, add the **-append** parameter:

```
Get-Process | Out-File C:\Test.txt -append
```

You can also use the MS-DOS redirection characters (`>` for write, `>>` for append) when using Windows PowerShell. This command writes data to the file `C:\Scripts\Test.txt`:

```
Get-Process > C:\Scripts\Test.txt
```

Another option is to use the **Export-CSV** cmdlet to save data as a comma-separated-values file:

```
Get-Process | Export-CSV C:\Test.csv
```

## How to Print Data

To print data to the default printer use the **Out-Printer** cmdlet:

```
Get-Process | Out-Printer
```

## How to Write Conditional Statements

To write an If statement use code similar to this:

```
$a = "white"
if ($a -eq "red")
    {"The color is red."}
elseif ($a -eq "white")
    {"The color is white."}
else
    {"The color is blue."}
```

Instead of writing a series of If statements you can use a **Switch** statement, which is equivalent to VBScript's **Select Case** statement:

```
$a = 2
switch ($a)
{
    1 {"The color is red."}
    2 {"The color is blue."}
    3 {"The color is green."}
    4 {"The color is yellow."}
    default {"Other."}
}
```

## How to Write For and For Each Loops

To write a For statement use code similar to this:

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

By comparison, a For Each statement might look like this:

```
foreach ($i in get-childitem c:\scripts)
    {$i.extension}
```

## How to Write Do Loops

To write a Do loop use code like the following, replacing the code between the curly braces with the code to be executed on each iteration of the loop. Oh: and replacing the code inside the parentheses with the loop condition:

```
$a = 1
do {$a; $a++}
while ($a -lt 10)
```

```
$a = 1
do {$a; $a++}
until ($a -gt 10)
```

# Windows PowerShell Quick Reference

## How to Create a COM Object

To work with a COM object use the **New-Object** cmdlet followed by the **-comobject** parameter and the appropriate ProgID:

```
$a = New-Object -comobject `
"Excel.Application"
$a.Visible = $True
```

## How to Create a .NET Object

To instantiate and use a .NET Framework object enclose the class name in square brackets, then separate the class name and the method using a pair of colons:

```
[system.Net.DNS]::resolve("207.46.198.30")
```

To create an object reference to a .NET Framework object use the **New-Object** cmdlet:

```
$a = new-object `
-type system.diagnostics.eventlog `
-argumentlist system
```

**Note.** This is a cursory overview of working with .NET. The two techniques shown here will not necessarily work with all .NET classes.

## How to Select Properties

To work with or display specified properties of a collection, pipe the returned results to the **Select-Object** cmdlet:

```
Get-Process | Select-Object Name, Company
```

## How to Sort Data

To sort data returned by Windows PowerShell simply pipe that data to the **Sort-Object** cmdlet, specifying the property you want to sort by:

```
Get-Process | Sort-Object ID
```

You can also add the **-descending** or **-ascending** parameters to specify a sort order:

```
Get-Process | Sort-Object ID -descending
```

You can even sort by multiple properties:

```
Get-Process | Sort-Object ProcessName, ID
```

## How to Work with WMI

To get computer information using WMI call the **Get-WMIObject** cmdlet followed by the class name:

```
Get-WMIObject Win32_BIOS
```

If the class you are interested in does not reside in the cimv2 namespace simply include the **-namespace** parameter:

```
Get-WMIObject SystemRestore `
-namespace root\default
```

To access data on another computer use the **-computername** parameter:

```
Get-WMIObject Win32_BIOS `
-computername atl-ws-01
```

To limit returned data, use a WQL query and the **-query** parameter:

```
Get-WMIObject -query `
"Select * From Win32_Service `
Where State = 'Stopped'"
```

## How to Bind to Active Directory

To bind to an Active Directory account use the LDAP provider:

```
$a = [adsis] "LDAP://cn=kenmyer, `
ou=Finance, dc=fabrikam, dc=com"
```

Listing all the objects in an OU is a little more complicated; however, one relatively easy way to accomplish this task is to bind to the OU and then use the **PSBase\_GetChildren()** method to retrieve a collection of items stored in that OU:

```
$objOU = [ADSI] `
"LDAP://ou=Finance, dc=fabrikam, dc=com"
$users = $objOU.PSBase.Get_Children()
$users | Select-Object displayName
```

## How to Bind to Local Accounts

To bind to a local account, use the WinNT provider:

```
$a = [adsis] "WinNT://atl-ws-01/kenmyer"
$a.FullName
```

# Windows PowerShell Quick Reference

## How to Get Help

To get complete help information for a Windows PowerShell cmdlet, use the **Get-Help** cmdlet along with the **-full** parameter. For example, to view the help information for the **Get-Process** cmdlet type the following:

```
Get-Help Get-Process -full
```

To view the example commands for a cmdlet use the **-examples** parameter:

```
Get-Help Get-Process -examples
```

If you can't remember the exact name for a cmdlet use **Get-Command** to retrieve a list of all the cmdlets available to you:

```
Get-Command
```

For a list of available aliases, use the **Get-Alias** cmdlet:

```
Get-Alias
```

## How to Change Security Settings

To run scripts from within Windows PowerShell you will need to change your security settings; by default, PowerShell only runs scripts signed by a trusted authority. To enable PowerShell to run all locally-created scripts (regardless of whether or not they have been signed) use the following command:

```
Set-ExecutionPolicy RemoteSigned
```

## How to "Interrogate" an Object

To get information about the properties and methods of an object retrieve an instance of that object and then "pipe" the object to the **Get-Member** cmdlet. For example, this command returns the properties and methods available when working with processes:

```
Get-Process | Get-Member
```

## How to Clear the Console Window

To clear the PowerShell window, use the **Clear-Host** function (or its alias, **cls**).

## How to Copy and Paste

To enable simple copying and pasting in the Windows PowerShell console do the following:

Start Windows PowerShell, then click the icon in the upper left-hand corner and choose **Properties**. In the **Windows PowerShell Properties** dialog box, on the **Options** tab, select **QuickEdit Mode** and then click OK.

To copy text in the console window select the text and then press ENTER. To paste text into the window click the right mouse button.

## How to Run a Script

To run a script from within Windows PowerShell, type the full path to the script (or type the script name if the script is stored in a folder that is part of your Windows path):

```
C:\Scripts\Test.ps1
```

If the path name includes blank spaces you must preface the path with an ampersand and enclose the path in double quotes. For example:

```
&"C:\Scripts\My Scripts\test.ps1"
```

From outside Windows PowerShell (e.g., from the **Run** dialog box or from a **Cmd.exe** window) you must call Windows PowerShell and then pass the script path as an argument to that call:

```
powershell.exe -noexit C:\Scripts\Test.ps1
```

The **-noexit** parameter ensures that the PowerShell window remains open after the script finishes running.

## How to Get More Information

For more information on writing Windows PowerShell scripts visit the TechNet Script Center at <http://technet.microsoft.com/en-us/scriptcenter/dd742419.aspx>.

