A shell is software that provides a customized interface designed for executing commands or scripts. (The term originated from OS nomenclature where the outer layer of a layered architecture OS was the interface between the user and the kernel of the OS.)

Most OS shells generally fall into one of two categories: command-line and graphical. Command-line shells provide a command-line interface (CLI) to the OS, while graphical shells provide a graphical user interface (GUI).

In either category the primary purpose of the shell is to invoke or “launch” other programs. In most modern environments, shells frequently have additional capabilities such as viewing the contents of directories.
Windows PowerShell is a command-line interface (CLI).

Two important features of PowerShell are scripts and cmdlets.

A script is a file of commands that is run when you execute or invoke the script.

A cmdlet (short for command-let, its pronounced like the long version) is a specialized (lightweight – they are technically instances of .NET Framework classes and are not stand-alone executables) command for completing common tasks in the PowerShell environment.

There are about 130 built-in cmdlets already defined in PowerShell and you can also define (create) your own custom cmdlets as well as import third-party cmdlets.
Windows PowerShell is particularly well suited for situations in which there are multiple servers and it is more efficient to manage them using a consistent set of scripts.

It is also ideal for managing servers with the Application Server role installed in situations where the applications need to be configured the same way and regular updates are applied.

Windows Server 2008 comes with PowerShell and it can be installed via the Server Manager (it also comes with Windows 7) and it can be easily downloaded into older server versions.
Some of the more common server administration tasks that can easily be handled through PowerShell include:

- Managing files and folders (directories).
- Managing network tasks.
- Managing fixed and removable storage devices.
- Configuring printing services.
- Managing software applications and updates.
- Managing Terminal Services.
- Managing server services and features.
- Managing Web server services
- Working with the system registry.
• PowerShell is not installed by default in Server 2008 (although it should be in Server 2008 R2 editions).

• To install PowerShell from the Server Manager:
  – Scroll down to the Features Summary.
  – Click Add Features.
  – Under Features, scroll to find Windows PowerShell and check its box.
  – Click Next and then click Install.
  – Click Close.
  – Close the Server Manager.

• The next few screen shots step you through this simple process.
Click Here
1. Check the Windows PowerShell checkbox.
2. Then click Next.
Click Install
Click Close after successful installation.
Back in Server Manager, the new feature lists under the installed features on this server.
Scripting – Windows PowerShell

• Once you’ve installed PowerShell on the server, you’re reading to take advantage of some of the cmdlets.

• With PowerShell installed, you should be able to find it on the server under the Start menu, click All Programs, click Accessories, Click Windows PowerShell, and Windows PowerShell should be there (see the next page).

  – Note: there will also be a Windows PowerShell ISE, which is the Integrated Scripting Environment. We’ll look at this later.

• Once you click on Windows PowerShell, you should see a screen like the one shown on page 14.
You’ll find PowerShell under the Accessories list from the Start, All Programs listing.
This is the default appearance with the window maximized. To change this setting see pages 43-44.
PS C:\users\Administrator> ~
• To view the files in the current folder (the default folder will be the Users/Administrator folder), one page of files at a time, enter the command:

    dir | more

• Press Enter after typing in the command (pressing the spacebar will give you the next page if there is one – probably not on our servers, since we don’t have much out there yet).

• What you’re doing here is executing the directory command and piping its output through to the more command which displays input one page at a time.

• The next page shows the execution of this command on one of my virtual servers.
```
PS C:\users\Administrator> dir | more

Directory: C:\users\Administrator

Mode         LastWriteTime          Length  Name
----------    ---------------        ------  ----------
-d---         10/3/2011 10:13 AM    .  .grasp_settings
-d---         11/16/2011 4:41 PM     .  .idlerc
-d---         3/15/2012 11:28 AM    .  .m2
-d---         2/22/2012 2:15 PM     d  Contacts
-d---         3/1/2012 8:21 AM      d  Desktop
-d---         2/22/2012 2:16 PM     d  Documents
-d---         9/21/2011 5:02 PM     d  Downloads
-d---         9/21/2011 5:02 PM     d  Favorites
-d---         9/21/2011 5:02 PM     d  Links
-d---         9/21/2011 5:02 PM     d  Music
-d---         11/16/2011 12:15 PM   d  MyScripts
-d---         9/21/2011 5:02 PM     d  Pictures
-d---         11/16/2011 4:30 PM     d  Python Scripts
-d---         9/21/2011 5:02 PM     d  Saved Games
-d---         9/21/2011 5:02 PM     d  Searches
-d---         9/21/2011 5:02 PM     d  Videos
-d---         9/21/2011 5:02 PM     d  workspace
-d---         11/29/2011 4:41 PM     d  workspace
```

To get a listing of the services currently running on your server, enter the command `get-service`, at the command prompt. A partial listing of the output of this cmdlet is shown on page 19.

To view a listing of all the currently defined cmdlets, enter the command `get-command | more`, at the command prompt. Here you will see the cmdlets one screen at a time, so press the spacebar to advance to the next screen. Simply repeat this until you’ve seen all the pages, or alternatively, press `q`, to quite and exit back to the command line if you don’t want to view all the pages. This command is illustrated on page 20.
get-service
PS C:\users\Administrator> get-command : more

<table>
<thead>
<tr>
<th>CommandType</th>
<th>Name</th>
<th>Definition</th>
</tr>
</thead>
<tbody>
<tr>
<td>Alias</td>
<td>%</td>
<td>ForEach-Object</td>
</tr>
<tr>
<td>Alias</td>
<td>?</td>
<td>Where-Object</td>
</tr>
<tr>
<td>Function</td>
<td>A</td>
<td>Set-Location A:</td>
</tr>
<tr>
<td>Alias</td>
<td>ac</td>
<td>Add-Content</td>
</tr>
<tr>
<td>Cmdlet</td>
<td>Add-Computer</td>
<td>Add-Computer [-DomainName] &lt;String&gt; [-Credenti</td>
</tr>
<tr>
<td>Cmdlet</td>
<td>Add-Content</td>
<td>t] &lt;Object&gt;</td>
</tr>
<tr>
<td>Cmdlet</td>
<td>Add-History</td>
<td>Add-History [[-InputObject] &lt;PSObject&gt;] [-Pa</td>
</tr>
<tr>
<td>Cmdlet</td>
<td>Add-Member</td>
<td>m -Add-Object [-Object] &lt;PSObject&gt; [-Include</td>
</tr>
<tr>
<td>Cmdlet</td>
<td>Add-PSSnapin</td>
<td>[-PassThru] -Add-Type [-TypeDefinition] &lt;Stri</td>
</tr>
<tr>
<td>Cmdlet</td>
<td>Add-Type</td>
<td>ng&gt; [-Language]</td>
</tr>
<tr>
<td>Alias</td>
<td>asnp</td>
<td>Add-PSSnapIn</td>
</tr>
<tr>
<td>Function</td>
<td>B</td>
<td>Set-Location B:</td>
</tr>
<tr>
<td>Alias</td>
<td>cat</td>
<td>Set-Location C:</td>
</tr>
<tr>
<td>Alias</td>
<td>cd</td>
<td>Get-Content</td>
</tr>
<tr>
<td>Alias</td>
<td>cd..</td>
<td>Set-Location .</td>
</tr>
<tr>
<td>Alias</td>
<td>cd\</td>
<td>Set-Location \</td>
</tr>
<tr>
<td>Alias</td>
<td>chdir</td>
<td>Set-Location ..</td>
</tr>
<tr>
<td>Cmdlet</td>
<td>Checkpoint-Computer</td>
<td>Checkpoint-Computer [-Description] &lt;String&gt; [</td>
</tr>
<tr>
<td>Alias</td>
<td>clc</td>
<td>Clear-Content</td>
</tr>
<tr>
<td>Alias</td>
<td>clear</td>
<td>Clear-Host</td>
</tr>
<tr>
<td>Cmdlet</td>
<td>Clear-Content</td>
<td>Clear-Content [-Path] &lt;String&gt; [-Filter &lt;Str</td>
</tr>
<tr>
<td>Cmdlet</td>
<td>Clear-EventLog</td>
<td>Clear-EventLog [-LogName] &lt;String&gt; [-EventLog</td>
</tr>
<tr>
<td>Cmdlet</td>
<td>Clear-History</td>
<td>Clear-History [[-Id] &lt;Int32&gt;] [[-Count] &lt;Int</td>
</tr>
<tr>
<td>Cmdlet</td>
<td>Clear-Item</td>
<td>Clear-Item [-Path] &lt;String&gt; [-Force] [-Filte</td>
</tr>
<tr>
<td>Cmdlet</td>
<td>Clear-ItemProperty</td>
<td>Clear-ItemProperty [-Path] &lt;String&gt; [-Name]</td>
</tr>
<tr>
<td>Cmdlet</td>
<td>Clear-Variable</td>
<td>Clear-Variable [-Name] &lt;String&gt; [-Include &lt;S</td>
</tr>
<tr>
<td>Alias</td>
<td>clhy</td>
<td>Clear-History</td>
</tr>
<tr>
<td>Alias</td>
<td>cl1</td>
<td>Clear-Item</td>
</tr>
<tr>
<td>Alias</td>
<td>clp</td>
<td>Clear-ItemProperty</td>
</tr>
<tr>
<td>Alias</td>
<td>cls</td>
<td>Clear-Host</td>
</tr>
<tr>
<td>Alias</td>
<td>clv</td>
<td>Clear-Variable</td>
</tr>
<tr>
<td>Alias</td>
<td>compare</td>
<td>Compare-Object</td>
</tr>
<tr>
<td>Cmdlet</td>
<td>Compare-Object</td>
<td>Compare-Object [-ReferenceObject] &lt;PSObject&gt; [</td>
</tr>
<tr>
<td>Cmdlet</td>
<td>Complete-Transaction</td>
<td>Complete-Transaction [-Verbose] [-Debug] [-Err</td>
</tr>
<tr>
<td>Cmdlet</td>
<td>Connect-WSMan</td>
<td>Connect-WSMan [-ComputerName] &lt;String&gt;] [-App</td>
</tr>
<tr>
<td>Cmdlet</td>
<td>ConvertFrom-Cs</td>
<td>ConvertFrom-Cs [-InputObject] &lt;PSObject&gt;] [</td>
</tr>
<tr>
<td>Cmdlet</td>
<td>ConvertFrom-Sec</td>
<td>ConvertFrom-SecureString [-SecureString] &lt;Secu</td>
</tr>
<tr>
<td>Cmdlet</td>
<td>ConvertFrom-StringData</td>
<td>ConvertFrom-StringData [-StringData] &lt;Strin</td>
</tr>
</tbody>
</table>

To direct input to this virtual machine, press Ctrl+G.
• One big plus of PowerShell is consistency. With many shells, the commands can vary in complexity; however, given the object-oriented nature of PowerShell, most cmdlets are fairly basic in their usage and are highly consistent.

• The power comes is using combinations of cmdlets.

• The cmdlets naming convention is for the first part to be a verb (for example, get-, format-, out-, or set-) that dictates what the cmdlet does (such as get information, format information, direct information, or set information).

• The next part is a noun, which specifies what is being acted on.
Everything is based around this verb-noun pair; for example, `get-process w*` retrieves information about processes whose names start with the letter w, as shown below.
Scripting – Windows PowerShell

- Although the output, as shown on the previous page, is tabular, this is not how the data is returned in PowerShell. It’s referenced in its .NET object format, but the default display format is a table.

- You can easily output in other formats, such as a list by piping the output of the `get-process` cmdlet to the `format-list` cmdlet.

```powershell
PS C:\users\Administrator> get-process w* | format-list
Id          : 544
Handles     : 100
CPU         : 0.3125
Name        : wininit

Id          : 576
Handles     : 120
CPU         : 0.796875
Name        : winlogon

Id          : 3976
Handles     : 85
CPU         : 0.046875
Name        : wuaclt
```
Scripting – Windows PowerShell

- Probably the greatest cmdlet (as well as the best verb-noun combination) that you’ll ever use is `get-help`.

- On its own, `get-help` gives you just basic information, but it can show you the names of other cmdlets, so you can detailed help on them.

- For example, `get-help format-*` will list all the cmdlets starting with `format-` to help you see the options available to you.
```bash
PS C:\users\Administrator> get-help format-*

Name                                                                 Category     Synopsis
----                                                                 ----     --------------------------
Format-List               Cmdlet     Formats the output as a list of properties in which each property appe
Format-Custom             Cmdlet     Uses a customized view to format the output.
Format-Table              Cmdlet     Formats the output as a table.
Format-Wide               Cmdlet     Formats objects as a wide table that displays only one property of eac
```
In addition to getting detailed help about a cmdlet, use the `get-help` command with the name of the cmdlet followed by `-detailed` to get all available help.

Add `-full` to just view a portion of the help, or add `-examples` to have examples of use listed for you.

Note that when the `-detailed` option is selected, the examples are also listed.

The following screen shots illustrates these cases. Note that the detailed case requires several pages of output and I only show the first one here. The same is often true for full and examples.
get-help format-list -detailed

NAME
Format-List

SYNOPSIS
Formats the output as a list of properties in which each property appears on a new line.

SYNTAX

DESCRIPTION
The Format-List cmdlet formats the output of a command as a list of properties in which each property is displayed on a separate line. You can use Format-List to format and display all or selected properties of an object as a list.

Because more space is available for each item in a list than in a table, Windows PowerShell displays more properties of the object in the list, and the property values are less likely to be truncated.

PARAMETERS
-DisplayError [[<SwitchParameter>]]
Displays errors at the command line.

-Expand <string>
Formats the collection object, as well as the objects in the collection. This parameter is designed to for objects that support the ICollection (System.Collections) interface. The default value is EnumOnly.

Valid values are:
--- EnumOnly: Displays the properties of the objects in the collection.
--- CoreOnly: Displays the properties of the collection object.
--- Both: Displays the properties of the collection object and the properties of objects in the collection.

-Force [[<SwitchParameter>]]
Directs the cmdlet to display all of the error information. Use with the DisplayError or ShowError parameter.
By default, when an error object is written to the error or display stream, only some of the error information is displayed.

-GroupBy <Object>
Formats the output in groups based on a shared property or value. Enter an expression or a property of the object.
get-help format-list -examples

NAME
Format-List

SYNOPSIS
Formats the output as a list of properties in which each property appears on a new line.

------------------------ EXAMPLE 1 ------------------------

C:\PS>get-service | format-list

Description

This command formats information about services on the computer as a list. By default, the services are formatted as a table. The Get-Service cmdlet gets objects representing the services on the computer. The pipeline operator passes the results through the pipeline to Format-List. Then, the Format-List command formats the service information in a list and sends it to the default output cmdlet for display.

------------------------ EXAMPLE 2 ------------------------

C:\PS>$a = get-childitem $pshome\*.ps1xml

Description

These commands display information about the PS1XML files in the Windows PowerShell directory as a list. The first command gets the objects representing the files and stores them in the $a variable. The second command uses Format-List to format information about objects stored in $a. This command uses the InputObject parameter to pass the objects to Format-List, which then sends the formatted output to the default output cmdlet for display.

------------------------ EXAMPLE 3 ------------------------

C:\PS>get-process | format-list -property name, basepriority, priorityclass

Description
We’ve already seen the cmdlet `get-command`. If you want to see all the commands that begin with a certain verb, such as `get`, issue the command `get-command -verb get`.

The output of this command is shown on the next page, but you might want to experiment a bit and try out some other options. For example, try listing all of the commands that use the verbs `add` or `new`. 
```powershell
get-command -verb get
```

<table>
<thead>
<tr>
<th>Cmdlet</th>
<th>Definition</th>
</tr>
</thead>
<tbody>
<tr>
<td>Get-Alias</td>
<td>Get-Alias [[-Name] &lt;String[]&gt;]</td>
</tr>
<tr>
<td>Get-AuthenticodeSignature</td>
<td>Get-AuthenticodeSignature [[-FilePath] &lt;String[…]]</td>
</tr>
<tr>
<td>Get-ChildItem</td>
<td>Get-ChildItem [[-Path] &lt;String[]&gt; [-Filter]]</td>
</tr>
<tr>
<td>Get-ComputerRestorePoint</td>
<td>Get-ComputerRestorePoint [[-RestorePoint]] [-ErrorAction] [-Debug]</td>
</tr>
<tr>
<td>Get-Content</td>
<td>Get-Content [[-Path] &lt;String[]&gt; [-ReadCount] [-InputObject]]</td>
</tr>
<tr>
<td>Get-Counter</td>
<td>Get-Counter [[-Counter] &lt;String[]&gt; [-SampleIn]]</td>
</tr>
<tr>
<td>Get-Credential</td>
<td>Get-Credential [[-Credential] &lt;PScredential&gt;]</td>
</tr>
<tr>
<td>Get-Date</td>
<td>Get-Date [[-DateTime] [-Year]]</td>
</tr>
<tr>
<td>Get-Event</td>
<td>Get-Event [[-SourceIdentifier] &lt;String[]&gt; [-Verb]]</td>
</tr>
<tr>
<td>Get-EventLog</td>
<td>Get-EventLog [[-LogName] &lt;String[]&gt; [[-Instanced]]</td>
</tr>
<tr>
<td>Get-EventSubscriber</td>
<td>Get-EventSubscriber [[-SourceIdentifier] [-Verb]]</td>
</tr>
<tr>
<td>Get-FormatData</td>
<td>Get-FormatData [[-TypeName] &lt;String[]&gt; [-Verb]]</td>
</tr>
<tr>
<td>Get-Help</td>
<td>Get-Help [[-Name] &lt;String[]&gt; [-Path]]</td>
</tr>
<tr>
<td>Get-Host</td>
<td>Get-Host [[-Id] &lt;Int64[]&gt; [-Count] &lt;Int32&gt;]</td>
</tr>
<tr>
<td>Get-HotFix</td>
<td>Get-HotFix [[-Id] &lt;String[]&gt; [-ComputerName] [-Path] [-Verb]]</td>
</tr>
<tr>
<td>Get-Item</td>
<td>Get-Item [[-Path] &lt;String[]&gt; [-Filter]]</td>
</tr>
<tr>
<td>Get-Job</td>
<td>Get-Job [[-Id] &lt;Int32[]&gt; [-Verb] [-Debug]]</td>
</tr>
<tr>
<td>Get-Lock</td>
<td>Get-Lock [[-PSProvider] &lt;String[]&gt; [-PSDrive]]</td>
</tr>
<tr>
<td>Get-Member</td>
<td>Get-Member [[-Name] &lt;String[]&gt; [-InputObject]]</td>
</tr>
<tr>
<td>Get-Module</td>
<td>Get-Module [[-Name] &lt;String[]&gt; [-All] [-Verb]]</td>
</tr>
<tr>
<td>Get-Process</td>
<td>Get-Process [[-Name] &lt;String[]&gt; [-ComputerName]]</td>
</tr>
<tr>
<td>Get-PSBreakpoint</td>
<td>Get-PSBreakpoint [[-Script] &lt;String[]&gt; [-Verb]]</td>
</tr>
<tr>
<td>Get-PSDrive</td>
<td>Get-PSDrive [[-Name] &lt;String[]&gt; [-Scope]]</td>
</tr>
<tr>
<td>Get-PSProvider</td>
<td>Get-PSProvider [[-PSProvider] &lt;String[]&gt; [-Verb]]</td>
</tr>
<tr>
<td>Get-PSSession</td>
<td>Get-PSSession [[-ComputerName] &lt;String[]&gt; [-Ue]]</td>
</tr>
<tr>
<td>Get-PSSessionConfiguration</td>
<td>Get-PSSessionConfiguration [[-Name] &lt;String[]&gt; [-Ue]]</td>
</tr>
<tr>
<td>Get-PSSnapin</td>
<td>Get-PSSnapin [[-Name] &lt;String[]&gt; [-Registered]]</td>
</tr>
<tr>
<td>Get-Service</td>
<td>Get-Service [[-Name] &lt;String[]&gt; [-ComputerName]]</td>
</tr>
<tr>
<td>Get-TraceSource</td>
<td>Get-TraceSource [[-Name] &lt;String[]&gt; [-Verb]]</td>
</tr>
</tbody>
</table>

To direct input to this virtual machine, press Ctrl+G.
Now that you’ve have some basic familiarity with PowerShell, let’s do something more useful with it… let’s try starting and stopping a process.

- What you might want to do before going any further is first run the `get-help -process` to list all the available commands that deal with a process. You should discover that there are five of these cmdlets.

What we’re going to do over the next few pages is start Notepad as a process running on our server and then use it and then stop the process. This will be illustrated by a sequence of screen shots from the server illustrating what is happening.

First off, we’ll see a screen shot of the current processes on the server. Notice that its alphabetically listed and Notepad is not running (Notepad++ is on my server).
Currently running processes do not include Notepad.
Enter the command: `start notepad` on the command line and press enter. Notepad immediately launches.
Reissue the command `get-process` and notice that now Notepad is listed.

Notice too in the tool tray that you can still see Notepad is there.
Notice on the previous screen shot that the id process id of the Notepad process was 3680. This is used in this version of the stop-process command to identify the process to be stopped.

Reissue the command get-process and notice that now Notepad is no longer listed.

Notice too in the tool tray that Notepad is no longer there.
You can also do a fair amount of customization of the PowerShell interface.

A common system administrator technique is to place scripts in a folder on a server that is frequently backed up. Thus, you might want PowerShell to open up in this default directory.

To illustrate doing this, let’s create a subdirectory in the C:\Users\Administrators folder named MyScripts. Then we’ll configure PowerShell to open in this folder.

To make some of these repetitive steps easier to accomplish, I also created a short-cut to PowerShell and put it on the start menu.
To set-up the default folder for PowerShell to open in, right click the short-cut to PowerShell and select Properties.

Locate the ShortCut tab on the Properties dialog box and in the Start in: text box enter the path to the new directory “C:\Users\Administrator\MyScripts”, then click OK.

Restart PowerShell and you should now see the new default directory loaded.
### Windows PowerShell Properties

<table>
<thead>
<tr>
<th>Compatibility</th>
<th>Security</th>
<th>Details</th>
<th>Previous Versions</th>
</tr>
</thead>
<tbody>
<tr>
<td>General</td>
<td>Shortcut</td>
<td>Options</td>
<td>Font</td>
</tr>
<tr>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
</tbody>
</table>

**Target type:** Application  
**Target location:** v1.0  
**Target:** `\WindowsPowerShell\v1.0\powershell.exe`

**Start in:** `c:\users\administrator\MyScripts`  
**Shortcut key:** None  
**Run:** Normal window  
**Comment:** Performs object-based (command-line) functions

---

To direct input to this virtual machine, press Ctrl+G.
PS C:\users\Administrator\MyScripts> -
You can also change the text size and the screen foreground and background colors and many other features including hot-keys and so on in PowerShell.

The next part simply shows you how to reset the text size and the screen colors to customize your PowerShell environment.

Again going through the desktop shortcut to PowerShell, right click on the short cut and select Properties. Locate the Font tab on the Properties dialog box and reset the Window size to 8x8 (the default is 8x12), then click OK.

Restart PowerShell and you should now see the new default screen size and font size for the window.
Administrator: Windows PowerShell

Copyright (C) 2009 Microsoft Corporation. All rights reserved.

PS C:\users\Administrator\MyScripts> -
• To change the screen colors for PowerShell, repeat the process but select the Colors tab.

• Again going through the desktop shortcut to PowerShell, right click on the short cut and select Properties. Locate the Colors tab on the Properties dialog box and reset the colors to your liking, then click OK.

• Restart PowerShell and you should now see the new colors appear.
Windows PowerShell

Copyright © 2009 Microsoft Corporation. All rights reserved.

PS C:\users\Administrator\MyScripts> _