How to do things: Windows Powershell

A quick and dirty guide to Windows Powershell for Linux/Bash users. Requires Powershell 5+.

“Beware that, when fighting monsters, you yourself do not become a monster... for when you gaze long into the abyss. The abyss gazes also into you.”

\- Nietzsche

Basics

PWSH 101

- Most built-in powershell commands return an Object, not text
- To see the type of a variable, use the `$var.getType()` method
- Variables must start with $ (but not if being used in `-OutVariable` arguments)
- To embed a command in another command, use ( )’s (just like Fish)
- Powershell has a man-like utility: `> Get-Help <command>`
- If you are lost about what command you need, try `> Get-Command <wildcard>`
- “Sudo” by opening a new pwsh window: `> Start-Process powershell -Verb RunAs`
- Powershell has great tab complete, even for command args

Jobs

A Powershell Job is similar to a Bash background process (&). You create a new job using the -AsJob argument on most commands. The command will then return a handle you can interact with. However, you can also get/mess with the jobs you started later.

The Invoke-Command can also manually start a job, even on other computers.

> Invoke-Command -ComputerName <name of domain-connected computer>
    -ScriptBlock { <powershell commands> }

For remote Powershell to work, the Enable-remoting command must have been run

View all of the started jobs with > Get-Job. Optionally, add the -Id parameter to just get one.

Manipulate the jobs with:
> Remove-Job
> Wait-Job
> Receive-Job # for when the job is done and you want it's data

Data Manipulation
This is where Powershell shines, since it has actual data types
$ head -n <n> => Select-Object -First <n>
$ tail -n <n> => Select-Object -Last <n>
$ sort => Sort-Object
$ uniq => Get-Unique
$ sed -i -e 's/<delimiter>/\n/g' => $string.Split("<delimiter>")
$ grep => $my_array | where {$_ -match "search string"}
The select <obj field> command can convert an array of objects into an array of string.

You can view a sortable GUI table of most commands by piping their output to Out-GridView (alias ogv). Note that this will not work on Server Core systems.

Base64
Convert to base64:
> [Convert]::ToBase64String([System.Text.Encoding]::Unicode.GetBytes("<MY TEXT>"))

What version of Powershell am I using?
> $PSVersionTable

What version of Windows am I using?
> [System.Environment]::OSVersion

What sort of computer am I using?
Will get information about the OS version, bios, and CPU:
> Get-ComputerInfo

Also for memory size:
> Get-Ciminstance Win32_OperatingSystem -Outvariable os
> $os.TotalVisibleMemorySize

Command History
> history # returns an array of commands you have run. Also use Ctrl+R to search it.

Clear history with:
> Clear-History

View Executables
> Get-command <exe name>
The get-command can also show the path of an exe

Path
View your path (where non-Powershell commands live):
> echo $env:Path

Weirdly, the $Path variable can also be used like a drive.
> cd Env; Get-content Path;

Unlike Linux, the Windows directories in $Path are separated by a “;”. You can add a directory to the $Path with:
> $Env:Path += ";<new dir>"

For Powershell module locations, see $Env:PSModulePath.

Files

Looking at Folders
> Get-ChildItem
Aliases: dir, ls

To look at drives (e.g. C:), try
> Get-PSDrive

To see raw storage devices, there is also
> Get-Disk

Searching for files
The Get-ChildItem command can do many of the same things as Unix find
> Get-Childitem -recurse -force -include *<FILENAME>*

You may want to include -ErrorAction SilentlyContinue since Windows has tons of restricted directories everywhere. You can pipe the search results in the Where-Object command for more advanced filtering. Other useful options are
- Exclude <name>
- File # show only files
- Directory # show only directories
- Hidden # show only? hidden files

View file hashes
> Get-FileHash
The -Algorithm argument can be used (with auto-complete) to select an algorithm. The default is SHA256.

Check file attributes
The NTFS file system supports a special way of including attributes with files called Alternate Data Streams (ADS). This is sometimes used by malware to store data secretly.

View the ADS attached to a file
> Get-Item -Path <file> -Streams *

View the content of an ADS
> Get-Content -Path <file> -Stream <Stream name>

Remove an ADS with
> Remote-Item -Path <file> -Stream <stream name>

Zip Files
Unzip a file
> Expand-Archive -LiteralPath <path to zip file> -DestinationPath .

Create a zip file
> Compress-Archive -Path <source folder> -DestinationPath <zip file>

Downloading Files from WWW
> Invoke-WebRequest <url>
HtmlWebResponseObject

Some useful options are
-OutFile # save the response as a file

The command automatically parses the resulting content if it is an html page and the “Internet Explorer Engine” is available. Some useful response fields are:

- .StatusCode
- .RawContent
- .Headers # a dictionary

Aliases: curl, wget

Names Pipes
A Pipe in Windows is a bit like a socket. Other computers can also communicate with them over SMB.

To list pipes:
> Get-ChildItem \\.\pipe\

ACL / File Permissions
The permissions of files or folders can be managed through powershell and ACL objects.

Step 1. Get the ACL object for a file or directory
> $acl = Get-acl <path to dir or file>

Step 2. Update the ACL object
First, you will most likely want to prevent the object from inheriting permissions from its parent directory, so that your changes are not overwritten.
> $acl.SetAccessRuleProtection($true, $false)
Or, you can remove every permission from the file (causing you to lose access) with

> $acl.SetAccessRuleProtection($true, $true)

To remove a specific user's permissions, try
> $rule = New-Object
System.Security.AccessControl.FileSystemAccessRule("<user>","ReadAndExecute","Deny")
> $acl.AddAccessRule($rule)

To add permission for a group, try
> $rule = New-Object
System.Security.AccessControl.FileSystemAccessRule("<group>","FullControl", Allow")
> $acl.AddAccessRule($rule)

Step 3. Set the ACL
> $acl | Set-acl <path to file or rir>

Block a file from executing
This can be done by removing the ReadAndExecute permission from the file.
> $acl = Get-acl <file>
> $acl.SetAccessRuleProtection($true, $false)
> New-Object
System.Security.AccessControl.FileSystemAccessRule("<user>","ReadAndExecute","Deny")
> $acl | set-acl <file>

Windows Registry

The windows registry is a database that stores settings for the system and users. The data is stored in .dat files in various places, but you will want to edit it through the powershell API. Registry data is exposed as a PSDrive, and can be edited like standard files.

View the subkeys in a key
> Get-ChildItem # (ls)
E.G. ls HKLM:\SYSTEM\Software\
Note that registry keys are themselves a collection of keys and values. You should be able to see all subkeys and values when you list the contents of a particular key.

Set a key
Figure out the name of the key, the name of the property, and the new value you want.
> Set-ItemProperty -Path <path> -Name <property name> -Value <new property value>

Delete a Key
> Remove-Item
To remove one key (which may contain many properties and values)
If there are full keys under the deleted key, you can remove the with the -Recurse path

Remove a single property from a key with

> Remove-ItemProperty -Path <key path> -Name <property name>

Create a Key
> New-Item -Path <key path, should probably be under HKLM or HCLU> -Name “key name”
Once you have created a key, you will want to add properties to it:
> New-ItemProperty -Path <key path> -Name <key name> -Value <the value of the property> -PropertyType <the property type, can be inferred, if in doubt try “String”>

You can also copy a key from a source to a destination with:
> Copy-Item # (cp) <source> <destination>

Networking

View network interface information
> Get-NetIPInterface
Returns an Array of CimInstance objects. Interfaces are separated by ip version/protocol. You will want to note the interface address for other networking commands.

DHCP
Active or inactive DHCP on an individual interface via
> Set-NetIpInterface -InterfaceIndex <index> -Dhcp <Enabled/Disabled>
> ipconfig /release
> ipconfig /renew

Set static Networking
First, set a static ip for an interface.
> New-NetIPaddress -InterfaceIndex <index> -IPAddress <address> -PrefixLength <CIDR prefix> -DefaultGateway <gateway address>

Next, set some DNS
> Set-DNSClientServerAddress -InterfaceIndex <index> -ServerAddresses <DNS1>,<DNS2>#...

View Networking Config
> Get-NetIPAddress
> Get-DNSClientServerAddress

Set Hostname
Check the hostname with `> hostname`
Set it to something new:
`> Rename-computer –computername “computer” –newname “newcomputername” –force –restart`

Warning: Changing hostname requires restart and may break AD integrations (especially if you try it on a DC).

Features and Packages

Powershell Modules
Powershell has modules that provide specific sets of functionality, such as administering AD servers. Users can easily create or install new modules. Generally, newer Powershell is smart about automatically importing modules if they are called.

To see all currently active modules, run: > Get-Module
By including the -ListAvailable argument, you can see all modules that are installed

Add Windows Features
These are the same as “Windows Optional Features”. They include different server components like AD and IIS.

Check the available features first:
```
> Get-WindowsFeatures
```

Add Windows Capabilities
Note that these are different than Windows features.

First, you may want to search for a capability (to get its exact name).
```
> Get-WindowsCapability -Online | ? Name -like <*wildcard*>
```

Once you have the name of the capability (including the version string at the end), you can install it:
```
> Add-WindowsCapability -Online -Name <Feature name>
```

You can always remove a capability/feature that you added previously:
```
Remove-WindowsCapability -Online -Name <name>
```

Some useful capabilities are:
OpenSSH.Client~~~~0.0.1.0
OpenSSH.Server~~~~0.0.1.0

Install Chocolatey
Start with an Admin Powershell

1. Save the Chocolatey install script
   > Invoke-WebRequest -OutFile ‘install.ps1’ ‘https://chocolatey.org/install.ps1’
2. Allow running scripts
   > Set-ExecutionPolicy Bypass -Scope Process -Force
3. > .\install.ps1

Services and Tasks

List services
> Get-Service -Include <optional wildcard>

View service description
> $svc = [System.Management.ManagementObject]::new("Win32_Service.Name='<svc>'")
> $svc.Description

Start a service
> Start-Service <service>

Enable a Service
> Set-Service -name <service name> -StartupType ‘Automatic’

Enable SSH
1. Check your version of Windows.
    a. > [System.Environment]::OSVersion
    b. Your version must be at least 10.0.17763.0 or everything will FAIL. If it is older, run updates. You also appear to need Hotfix KB4519569
2. Add the sshd component
    a. > Add-WindowsCapability -Online -Name OpenSSH.Server~~~~0.0.1.0
3. Start the service
    a. Start-Service sshd
4. Enable the service
    a. > Set-Service -name sshd -StartupType ‘Automatic’
5. You should be able to log on with password auth from another computer.

The config file for sshd is at $env:ProgramData\ssh\sshd_config. If you need to edit it over SSH, install vim with choco. Set the default shell to powershell with:
> New-ItemProperty -Path "HKLM:\SOFTWARE\OpenSSH" -Name DefaultShell -Value "C:\Windows\System32\WindowsPowerShell\v1.0\powershell.exe" -PropertyType String -Force

Enable RDP
1. Set the “fDenyTSConnections” registry key to false
    a. `> Set-ItemProperty -Path 'HKLM:\System\CurrentControlSet\Control\Terminal Server' -name "fDenyTSConnections" -value 0`
2. Allow RDP through the firewall
    a. `> Enable-NetFirewallRule -DisplayGroup ‘Remote Desktop’`

Note that this will allow RDP connections from anywhere. You probably want to limit them to specific hosts.

You can also quickly disable rdp:
`> Disable-NetFirewallRule -DisplayGroup “Remote Desktop”`

Startup Commands
When programs are set to start on startup, an instance of the Win32_StartupCommand CIM class is created.

> Get-CimInstance -ClassName Win32_StartupCommand

Sysinternals is recommended to fully manage startup programs.

Scheduled Task Listing
A scheduled task in Windows is like a more powerful cron job. They can be set to fire at certain times, on regular intervals, or in response to certain events. For a GUI version, run taskschd.msc.

The basic command to list is:
```
> Get-ScheduledTask
```
To filter tasks, use the TaskName or TaskPath arguments

For better info viewing, here is a “nice” one-liner
```
> Get-ScheduledTask | where {$_.State -eq "Ready"} | select TaskName,
{$_.Actions.Execute}, {$_Actions.Arguments}, author, taskpath, Description
```

If there are scheduled tasks you do not want to execute, try Disable-ScheduledTask or Unregister-ScheduledTask.

Scheduled Task Creation
There are three steps here. First, create a “Task Action”
```
> $action = New-ScheduledTaskAction -Execute ‘<exe>’
      -Arguments “<optional exe args>”
```
Then, create a “Task Trigger” to determine when to run the task. Here are some examples
```
      > $trigger = New-ScheduledTaskTrigger -Daily -At 9am
      > $trigger = New-ScheduledTaskTrigger -Once -At 3am
      > $trigger = New-ScheduledTaskTrigger -AtLogon -User <User>
```
Finally, you can create the task itself
```
> Register-ScheduledTask -Action $action -Trigger $trigger -TaskName "<name>"
```

Users and Groups

List local users
> Get-LocalUser
Note that domain controllers don’t really have local users. All domain users exist on them.

List local groups
> Get-LocalGroup

Or, to see members:
> Get-LocalGroupMembers <group name>

Add local user
> New-LocalUser -Name <username> -Password (Read-Host -AsSecureString)
Some other optional flags may be -Description and -FullName, which both accept strings

Change local users password
> Set-LocalUser -Name <name> -Password (Read-Host -AsSecureString)

Then type the password and hit Enter

Disable a user
> Disable-localUser -Name <name>
And the inverse...
> Enable-LocalUser -Name <name>
You can pipe in objects from the Get-LocalUser command as well.

For AD commands, see [All About AD]

Other

Power
Restart:
> Restart-Computer
Add optional computer names as arguments in an AD domain

Shutdown:
> shutdown.exe /p

Dialog Box
Sometimes, you just gotta show a message
>$wshell = New-Object -ComObject Wscript.Shell
>$wshell.Popup("<your message>",0,"Done",0x1)

Resources

https://docs.microsoft.com/en-us/windows-server/administration/server-core/server-core-administer