| **Command** | **Description** | **Options** | **Examples** |
| --- | --- | --- | --- |
| [ls](https://www.geeksforgeeks.org/ls-command-in-linux/) | List files and directories. | * **-l**: Long format listing. * **-a**: Include hidden files hidden ones * **-h**: Human-readable file sizes. | * **ls -l** displays files and directories with detailed information. * **ls -a** shows all files and directories, including * **ls -lh**  displays file sizes in a human-readable format. |
| [cd](https://www.geeksforgeeks.org/cd-command-in-linux-with-examples/) | Change directory. |  | * **cd /path/to/directory** changes the current directory to the specified path. |
| [pwd](https://www.geeksforgeeks.org/pwd-command-in-linux-with-examples/) | Print current working directory. |  | * **pwd** displays the current working directory. |
| [mkdir](https://www.geeksforgeeks.org/mkdir-command-in-linux-with-examples/) | Create a new directory. |  | * **mkdir my\_directory** creates a new directory named "my\_directory". |
| [rm](https://www.geeksforgeeks.org/rm-command-linux-examples/) | Remove files and directories. | * **-r**: Remove directories recursively. * **-f**: Force removal without confirmation. | * **rm file.txt** deletes the file named "file.txt". * **rm -r my\_directory** deletes the directory "my\_directory" and its contents. * **rm -f file.txt** forcefully deletes the file "file.txt" without confirmation. |
| [cp](https://www.geeksforgeeks.org/cp-command-linux-examples/) | Copy files and directories. | * **-r**: Copy directories recursively. | * **cp -r directory destination** copies the directory "directory" and its contents to the specified destination. * **cp file.txt destination** copies the file "file.txt" to the specified destination. |
| [**mv**](https://www.geeksforgeeks.org/mv-command-linux-examples/) | Move/rename files and directories. |  | * **mv file.txt new\_name.txt**  renames the file "file.txt" to "new\_name.txt". * **mv file.txt directory** moves the file "file.txt" to the specified directory. |
| [touch](https://www.geeksforgeeks.org/touch-command-in-linux-with-examples/) | Create an empty file or update file timestamps. |  | * **touch file.txt**  creates an empty file named "file.txt". |
| [cat](https://www.geeksforgeeks.org/cat-command-in-linux-with-examples/) | View the contents of a file. |  | * **cat file.txt**  displays the contents of the file "file.txt". |
| [head](https://www.geeksforgeeks.org/head-command-linux-examples/) | Display the first few lines of a file. | * **-n**: Specify the number of lines to display. | * **head file.txt**  shows the first 10 lines of the file "file.txt". * **head -n 5 file.txt**  displays the first 5 lines of the file "file.txt". |
| [tail](https://www.geeksforgeeks.org/tail-command-linux-examples/) | Display the last few lines of a file. | * **-n**: Specify the number of lines to display. | * **tail file.txt**  shows the last 10 lines of the file "file.txt". * **tail -n 5 file.txt** displays the last 5 lines of the file "file.txt". |
| [ln](https://www.geeksforgeeks.org/ln-command-in-linux-with-examples/) | Create links between files. | * **-s**: Create symbolic (soft) links. | * **ln -s source\_file link\_name**  creates a symbolic link named "link\_name" pointing to "source\_file". |
| [find](https://www.geeksforgeeks.org/find-command-in-linux-with-examples/) | Search for files and directories. | * **-name**: Search by filename. * **-type**: Search by file type. | * **find /path/to/search -name "\*.txt"**  searches for all files with the extension ".txt" in the specified directory. |

**2. File Permission Commands**

File permissions on Linux and Unix systems control access to files and directories. There are three basic permissions: read, write, and execute. Each permission can be granted or denied to three different categories of users: the owner of the file, the members of the file's group, and everyone else.

Here are some file permission commands:

| **Command** | **Description** | **Options** | **Examples** |
| --- | --- | --- | --- |
| [chmod](https://www.geeksforgeeks.org/chmod-command-linux/) | Change file permissions. | * **u**: User/owner permissions. * **g**: Group permissions. * **o**: Other permissions. * **+**: Add permissions. * **-**: Remove permissions. * **=**: Set permissions explicitly. | * **chmod u+rwx file.txt**  grants read, write, and execute permissions to the owner of the file. |
| [chown](https://www.geeksforgeeks.org/chown-command-in-linux-with-examples/) | Change file ownership. |  | * **chown user file.txt** changes the owner of "file.txt" to the specified user. |
| [chgrp](https://www.geeksforgeeks.org/chgrp-command-in-linux-with-examples/) | Change group ownership. |  | * **chgrp group file.txt** changes the group ownership of "file.txt" to the specified group. |
| [umask](https://www.geeksforgeeks.org/umask-command-in-linux-with-examples/) | Set default file permissions. |  | * **umask 022**  sets the default file permissions to read and write for the owner, and read-only for group and others. |

**3. File Compression and Archiving Commands**

Here are some file compression and archiving commands in Linux:

| **Commands** | **Description** | **Options** | **Examples** |
| --- | --- | --- | --- |
| [tar](https://www.geeksforgeeks.org/tar-command-linux-examples/) | Create or extract archive files. | * **-c**: Create a new archive. * **-x**: Extract files from an archive. * **-f**: Specify the archive file name. * **-v**: Verbose mode. * **-z**: Compress the archive with gzip. * **-j**: Compress the archive with bzip2. | * **tar -czvf archive.tar.gz files/**  creates a compressed tar archive named "archive.tar.gz" containing the files in the "files/" directory. |
| [gzip](https://www.geeksforgeeks.org/gzip-command-linux/) | Compress files. | * **-d**: Decompress files. | * **gzip file.txt** compresses the file "file.txt" and renames it as "file.txt.gz". |
| [zip](https://www.geeksforgeeks.org/zip-command-in-linux-with-examples/) | Create compressed zip archives. | * **-r**: Recursively include directories. | * **zip archive.zip file1.txt file2.txt**  creates a zip archive named "archive.zip" containing "file1.txt" and "file2.txt". |

**4. Process Management Commands**

In Linux, process management commands allow you to monitor and control running processes on the system. Here are some commonly used process management commands:

| **Commands** | **Description** | **Options** | **Examples** |
| --- | --- | --- | --- |
| [ps](https://www.geeksforgeeks.org/ps-command-in-linux-with-examples/) | Display running processes. | * **-aux**: Show all processes. | * **ps aux**  shows all running processes with detailed information. |
| [top](https://www.geeksforgeeks.org/top-command-in-linux-with-examples/) | Monitor system processes in real-time. |  | * **top** displays a dynamic view of system processes and their resource usage. |
| [kill](https://www.geeksforgeeks.org/kill-command-in-linux-with-examples/) | Terminate a process. | * **-9**: Forcefully kill a process. | * **kill PID**  terminates the process with the specified process ID. |
| [pkill](https://www.geeksforgeeks.org/kill-command-in-linux-with-examples/) | Terminate processes based on their name. |  | * **pkill process\_name**  terminates all processes with the specified name. |
| **pgrep** | List processes based on their name. |  | * **pgrep process\_name**  lists all processes with the specified name. |
| [grep](https://www.geeksforgeeks.org/grep-command-in-unixlinux/) | used to search for specific patterns or regular expressions in text files or streams and display matching lines. | * **-i**: Ignore case distinctions while searching. * **-v**: Invert the match, displaying non-matching lines. * **-r or -R**: Recursively search directories for matching patterns. * **-l**: Print only the names of files containing matches. * **-n**: Display line numbers alongside matching lines. * **-w**: Match whole words only, rather than partial matches. * **-c**: Count the number of matching lines instead of displaying them. * **-e**: Specify multiple patterns to search for. * **-A**: Display lines after the matching line. * **-B**: Display lines before the matching line. * **-C**: Display lines both before and after the matching line. | * **grep -i "hello" file.txt** * **grep -v "error" file.txt** * **grep -r "pattern" directory/** * **grep -l "keyword" file.txt** * **grep -n "pattern" file.txt** In these examples we are extracting our desirec output from filename (file.txt) |

**5. System Information Commands**

In Linux, there are several commands available to gather system information. Here are some commonly used system information commands:

| **sudCommand** | **Description** | **Options** | **Examples** |
| --- | --- | --- | --- |
| [uname](https://www.geeksforgeeks.org/uname-command-in-linux-with-examples/) | Print system information. | * **-a**: All system information. | * **uname -a** displays all system information. |
| [whoami](https://www.geeksforgeeks.org/whoami-command-linux-example/) | Display current username. |  | * **whoami**  shows the current username. |
| [df](https://www.geeksforgeeks.org/df-command-in-linux-with-examples/) | Show disk space usage. | * **-h**: Human-readable sizes. | * **df -h**  displays disk space usage in a human-readable format. |
| [du](https://www.geeksforgeeks.org/du-command-linux-examples/) | Estimate file and directory sizes. | * **-h**: Human-readable sizes. * **-s**: Display total size only. | * **du -sh directory/**  provides the total size of the specified directory. |
| [free](https://www.geeksforgeeks.org/free-command-linux-examples/) | Display memory usage information. | * **-h**: Human-readable sizes. | * **free -h**  displays memory usage in a human-readable format. |
| [uptime](https://www.geeksforgeeks.org/linux-uptime-command-with-examples/) | Show system uptime. |  | * **uptime** shows the current system uptime. |
| **lscpu** | Display CPU information. |  | * **lscpu**  provides detailed CPU information. |
| **lspci** | List PCI devices. |  | * **lspci** List PCI devices. |
| [lsusb](https://www.geeksforgeeks.org/lsusb-command-in-linux-with-examples/) | List USB devices. |  | * **lsusb**  lists all connected USB devices. |

**6. Networking Commands**

In Linux, there are several networking commands available to manage and troubleshoot network connections. Here are some commonly used networking commands:

| **Command** | **Description** | **Examples** |
| --- | --- | --- |
| [ifconfig](https://www.geeksforgeeks.org/ifconfig-command-in-linux-with-examples/) | Display network interface information. | * **ifconfig**  shows the details of all network interfaces. |
| [ping](https://www.geeksforgeeks.org/ping-command-in-linux-with-examples/) | Send ICMP echo requests to a host. | * **ping google.com**  sends ICMP echo requests to "google.com" to check connectivity. |
| [netstat](https://www.geeksforgeeks.org/netstat-command-linux/) | Display network connections and statistics. | * **netstat -tuln**  shows all listening TCP and UDP connections. |
| **ss** | Display network socket information. | * **ss -tuln**  shows all listening TCP and UDP connections. |
| [ssh](https://www.geeksforgeeks.org/ssh-command-in-linux-with-examples/) | Securely connect to a remote server. | * **ssh user@hostname**  initiates an SSH connection to the specified hostname. |
| [scp](https://www.geeksforgeeks.org/scp-command-in-linux-with-examples/) | Securely copy files between hosts. | * **scp file.txt user@hostname:/path/to/destination**  securely copies "file.txt" to the specified remote host. |
| [wget](https://www.geeksforgeeks.org/wget-command-in-linux-unix/) | Download files from the web. | * **wget http://example.com/file.txt**  downloads "file.txt" from the specified URL. |
| [curl](https://www.geeksforgeeks.org/curl-command-in-linux-with-examples/) | Transfer data to or from a server. | * **curl http://example.com**  retrieves the content of a webpage from the specified URL. |

**7. IO Redirection Commands**

In Linux, IO (Input/Output) redirection commands are used to redirect the standard input, output, and error streams of commands and processes. Here are some commonly used IO redirection commands:

| **Command** | **Description** |
| --- | --- |
| cmd < file | Input of cmd is taken from file. |
| cmd > file | Standard output (stdout) of cmd is redirected to file. |
| cmd 2> file | Error output (stderr) of cmd is redirected to file. |
| cmd 2>&1 | stderr is redirected to the same place as stdout. |
| cmd1 <(cmd2) | Output of cmd2 is used as the input file for cmd1. |
| cmd > /dev/null | Discards the stdout of cmd by sending it to the null device. |
| cmd &> file | Every output of cmd is redirected to file. |
| cmd 1>&2 | stdout is redirected to the same place as stderr. |
| cmd >> file | Appends the stdout of cmd to file. |

**8. Environment Variable Commands**

In Linux, environment variables are used to store configuration settings, system information, and other variables that can be accessed by processes and shell scripts. Here are some commonly used environment variable commands:

| **Command** | **Description** |
| --- | --- |
| **export VARIABLE\_NAME=value** | Sets the value of an environment variable. |
| **echo $VARIABLE\_NAME** | Displays the value of a specific environment variable. |
| **env** | Lists all environment variables currently set in the system. |
| **unset VARIABLE\_NAME** | Unsets or removes an environment variable. |
| **export -p** | Shows a list of all currently exported environment variables. |
| **env VAR1=value COMMAND** | Sets the value of an environment variable for a specific command. |
| **printenv** | Displays the values of all environment variables. |

**9. User Management Commands**

In Linux, user management commands allow you to create, modify, and manage user accounts on the system. Here are some commonly used user management commands:

| **Command** | **Description** |
| --- | --- |
| **who** | Show who is currently logged in. |
| **sudo adduser username** | Create a new user account on the system with the specified username. |
| **finger** | Display information about all the users currently logged into the system, including their usernames, login time, and terminal. |
| **sudo deluser USER GROUPNAME** | Remove the specified user from the specified group. |
| **last** | Show the recent login history of users. |
| **finger username** | Provide information about the specified user, including their username, real name, terminal, idle time, and login time. |
| **sudo userdel -r username** | Delete the specified user account from the system, including their home directory and associated files. The -r option ensures the removal of the user's files. |
| **sudo passwd -l username** | Lock the password of the specified user account, preventing the user from logging in. |
| **su - username** | Switch to another user account with the user's environment. |
| **sudo usermod -a -G GROUPNAME USERNAME** | Add an existing user to the specified group. The user is added to the group without removing them from their current groups. |

**10. Shortcuts Commands**

There are many shortcuts commands in Linux that can help you be more productive. Here are a few of the most common ones:

**10.1: Bash Shortcuts Commands:**

| **Navigation** | **Description** | **Editing** | **Description** | **History** | **Description** |
| --- | --- | --- | --- | --- | --- |
| **Ctrl + A** | Move to the beginning of the line. | **Ctrl + U** | Cut/delete from the cursor position to the beginning of the line. | **Ctrl + R** | Search command history (reverse search). |
| **Ctrl + E** | Move to the end of the line. | **Ctrl + K** | Cut/delete from the cursor position to the end of the line. | **Ctrl + G** | Escape from history search mode. |
| **Ctrl + B** | Move back one character. | **Ctrl + W** | Cut/delete the word before the cursor. | **Ctrl + P** | Go to the previous command in history. |
| **Ctrl + F** | Move forward one character. | **Ctrl + Y** | Paste the last cut text. | **Ctrl + N** | Go to the next command in history. |
| **Alt + B** | Move back one word | **Ctrl + L** | Clear the screen. | **Ctrl + C** | Terminate the current command. |
| **Alt + F** | Move forward one word. |  |  |  |  |

Navigating the File System File Manipulation cd [directory] pwd ls [options] [directory] mkdir [directory] rmdir [directory] cp [source] [destination] mv [source] [destination] Change directory Print working directory List directory contents Create a new directory Remove a directory Copy files or directories cat [file] head [options] [file] tail [options] [file] less [file] grep [pattern] [file] Output the contents of a file Output the first lines of a file Output the last lines of a file View the contents of a file interactively Search for a pattern in a file wc [options] [file] Count the number of lines, words, or characters in a file Move or rename files or directories rm [options] [file] touch [file] Remove files or directories Create an empty file Permissions chmod [permissions] [file] Change the permissions of a file or directory chown [user:group] [file] Archiving and Compression tar [options] [files/directories] gzip [file] gunzip [file.gz] Create or extract tar archives Compress a file Decompress a gzipped file chgrp [group] [file] umask [mask] Change the owner and group of a file or directory Change the group of a file or directory Set the default file permissions for newly created files Process Management zip [archive.zip] [files/directories] unzip [archive.zip] Create a zip archive Extract files from a zip archive ps [options] kill [process\_ID] Display information about active processes Terminate a process top bg [job\_ID] fg [job\_ID] Display and manage the top processes Move a job to the background Bring a background job to the foreground

| **SNo.** | **Windows** | **Linux** | **Description** |
| --- | --- | --- | --- |
| 1. | dir | ls -l | Directory listing |
| 2. | ren | mv | Rename a file |
| 3. | copy | cp | Copying a file |
| 4. | move | mv | Moving a file |
| 5. | cls | clear | Clear Screen |
| 6. | del | rm | Delete file |
| 7. | fc | diff | Compare contents of files |
| 8. | find | grep | Search for a string in a file |
| 9. | command /? | man command | Display the manual/help details of the command |
| 10. | chdir | pwd | Returns your current directory location |
| 11. | time | date | Displays the time |
| 12. | cd | cd | Change the current directory |
| 13. | md | mkdir | To create a new directory/folder |
| 14. | echo | echo | To print something on the screen |
| 15. | edit | vim(depends on editor) | To write in to files. |
| 16. | exit | exit | To leave the terminal/command window. |
| 17. | format | mke2fs or mformat | To format a drive/partition. |
| 18. | free | mem | To display free space. |
| 19. | rmdir | rm -rf/rmdir | To delete a directory. |
| 20. | taskkill | kill | To kill a task. |
| 21. | tasklist | ps x | To list running tasks. |
| 22. | set var=value | export var=value | To set environment variables. |
| 23. | attrib | chown/chmod | To change file permissions. |
| 24. | tracert | traceroute | To print the route packets trace to network host. |
| 25. | at | cron | daemon to execute scheduled commands. |
| 26. | type | cat | To print contents of a file. |
| 27. | ping | ping | To send ICMP ECHO\_REQUEST to network hosts. |
| 28. | nslookup | nslookup | To query Internet name servers interactively. |
| 29. | chdisk | du -s | For disk usage. |
| 30. | tree | ls -R | To list directory recursively. |

|  |
| --- |
| Get-Command # Retrieves a list of all the commands available to PowerShell |
|  | # (native binaries in $env:PATH + cmdlets / functions from PowerShell modules) |
|  | Get-Command -Module Microsoft\* # Retrieves a list of all the PowerShell commands exported from modules named Microsoft\* |
|  | Get-Command -Name \*item # Retrieves a list of all commands (native binaries + PowerShell commands) ending in "item" |
|  |  |
|  | Get-Help # Get all help topics |
|  | Get-Help -Name about\_Variables # Get help for a specific about\_\* topic (aka. man page) |
|  | Get-Help -Name Get-Command # Get help for a specific PowerShell function |
|  | Get-Help -Name Get-Command -Parameter Module # Get help for a specific parameter on a specific command |
|  |  |
|  |  |
|  | ################################################### |
|  | # Operators |
|  | ################################################### |
|  |  |
|  | $a = 2 # Basic variable assignment operator |
|  | $a += 1 # Incremental assignment operator |
|  | $a -= 1 # Decrement assignment operator |
|  |  |
|  | $a -eq 0 # Equality comparison operator |
|  | $a -ne 5 # Not-equal comparison operator |
|  | $a -gt 2 # Greater than comparison operator |
|  | $a -lt 3 # Less than comparison operator |
|  |  |
|  | $FirstName = 'Trevor' |
|  | $FirstName -like 'T\*' # Perform string comparison using the -like operator, which supports the wildcard (\*) character. Returns $true |
|  |  |
|  | $BaconIsYummy = $true |
|  | $FoodToEat = $BaconIsYummy ? 'bacon' : 'beets' # Sets the $FoodToEat variable to 'bacon' using the ternary operator |
|  |  |
|  | 'Celery' -in @('Bacon', 'Sausage', 'Steak', 'Chicken') # Returns boolean value indicating if left-hand operand exists in right-hand array |
|  | 'Celery' -notin @('Bacon', 'Sausage', 'Steak') # Returns $true, because Celery is not part of the right-hand list |
|  |  |
|  | 5 -is [string] # Is the number 5 a string value? No. Returns $false. |
|  | 5 -is [int32] # Is the number 5 a 32-bit integer? Yes. Returns $true. |
|  | 5 -is [int64] # Is the number 5 a 64-bit integer? No. Returns $false. |
|  | 'Trevor' -is [int64] # Is 'Trevor' a 64-bit integer? No. Returns $false. |
|  | 'Trevor' -isnot [string] # Is 'Trevor' NOT a string? No. Returns $false. |
|  | 'Trevor' -is [string] # Is 'Trevor' a string? Yes. Returns $true. |
|  | $true -is [bool] # Is $true a boolean value? Yes. Returns $true. |
|  | $false -is [bool] # Is $false a boolean value? Yes. Returns $true. |
|  | 5 -is [bool] # Is the number 5 a boolean value? No. Returns $false. |
|  |  |
|  | ################################################### |
|  | # Regular Expressions |
|  | ################################################### |
|  |  |
|  | 'Trevor' -match '^T\w\*' # Perform a regular expression match against a string value. # Returns $true and populates $matches variable |
|  | $matches[0] # Returns 'Trevor', based on the above match |
|  |  |
|  | @('Trevor', 'Billy', 'Bobby') -match '^B' # Perform a regular expression match against an array of string values. Returns Billy, Bobby |
|  |  |
|  | $regex = [regex]'(\w{3,8})' |
|  | $regex.Matches('Trevor Bobby Dillon Joe Jacob').Value # Find multiple matches against a singleton string value. |
|  |  |
|  | ################################################### |
|  | # Flow Control |
|  | ################################################### |
|  |  |
|  | if (1 -eq 1) { } # Do something if 1 is equal to 1 |
|  |  |
|  | do { 'hi' } while ($false) # Loop while a condition is true (always executes at least once) |
|  |  |
|  | while ($false) { 'hi' } # While loops are not guaranteed to run at least once |
|  | while ($true) { } # Do something indefinitely |
|  | while ($true) { if (1 -eq 1) { break } } # Break out of an infinite while loop conditionally |
|  |  |
|  | for ($i = 0; $i -le 10; $i++) { Write-Host $i } # Iterate using a for..loop |
|  | foreach ($item in (Get-Process)) { } # Iterate over items in an array |
|  |  |
|  | switch ('test') { 'test' { 'matched'; break } } # Use the switch statement to perform actions based on conditions. Returns string 'matched' |
|  | switch -regex (@('Trevor', 'Daniel', 'Bobby')) { # Use the switch statement with regular expressions to match inputs |
|  | 'o' { $PSItem; break } # NOTE: $PSItem or $\_ refers to the "current" item being matched in the array |
|  | } |
|  | switch -regex (@('Trevor', 'Daniel', 'Bobby')) { # Switch statement omitting the break statement. Inputs can be matched multiple times, in this scenario. |
|  | 'e' { $PSItem } |
|  | 'r' { $PSItem } |
|  | } |
|  |  |
|  | ################################################### |
|  | # Variables |
|  | ################################################### |
|  |  |
|  |  |
|  | $a = 0 # Initialize a variable |
|  | [int] $a = 'Trevor' # Initialize a variable, with the specified type (throws an exception) |
|  | [string] $a = 'Trevor' # Initialize a variable, with the specified type (doesn't throw an exception) |
|  |  |
|  | Get-Command -Name \*varia\* # Get a list of commands related to variable management |
|  |  |
|  | Get-Variable # Get an array of objects, representing the variables in the current and parent scopes |
|  | Get-Variable | ? { $PSItem.Options -contains 'constant' } # Get variables with the "Constant" option set |
|  | Get-Variable | ? { $PSItem.Options -contains 'readonly' } # Get variables with the "ReadOnly" option set |
|  |  |
|  | New-Variable -Name FirstName -Value Trevor |
|  | New-Variable FirstName -Value Trevor -Option Constant # Create a constant variable, that can only be removed by restarting PowerShell |
|  | New-Variable FirstName -Value Trevor -Option ReadOnly # Create a variable that can only be removed by specifying the -Force parameter on Remove-Variable |
|  |  |
|  | Remove-Variable -Name firstname # Remove a variable, with the specified name |
|  | Remove-Variable -Name firstname -Force # Remove a variable, with the specified name, that has the "ReadOnly" option set |
|  |  |
|  | ################################################### |
|  | # Functions |
|  | ################################################### |
|  |  |
|  | function add ($a, $b) { $a + $b } # A basic PowerShell function |
|  |  |
|  | function Do-Something { # A PowerShell Advanced Function, with all three blocks declared: BEGIN, PROCESS, END |
|  | [CmdletBinding]()] |
|  | param () |
|  | begin { } |
|  | process { } |
|  | end { } |
|  | } |
|  |  |
|  | ################################################### |
|  | # Working with Modules |
|  | ################################################### |
|  |  |
|  | Get-Command -Name \*module\* -Module mic\*core # Which commands can I use to work with modules? |
|  |  |
|  | Get-Module -ListAvailable # Show me all of the modules installed on my system (controlled by $env:PSModulePath) |
|  | Get-Module # Show me all of the modules imported into the current session |
|  |  |
|  | $PSModuleAutoLoadingPreference = 0 # Disable auto-loading of installed PowerShell modules, when a command is invoked |
|  |  |
|  | Import-Module -Name NameIT # Explicitly import a module, from the specified filesystem path or name (must be present in $env:PSModulePath) |
|  | Remove-Module -Name NameIT # Remove a module from the scope of the current PowerShell session |
|  |  |
|  | New-ModuleManifest # Helper function to create a new module manifest. You can create it by hand instead. |
|  |  |
|  | New-Module -Name trevor -ScriptBlock { # Create an in-memory PowerShell module (advanced users) |
|  | function Add($a,$b) { $a + $b } } |
|  |  |
|  | New-Module -Name trevor -ScriptBlock { # Create an in-memory PowerShell module, and make it visible to Get-Module (advanced users) |
|  | function Add($a,$b) { $a + $b } } | Import-Module |
|  |  |
|  | ################################################### |
|  | # Module Management |
|  | ################################################### |
|  |  |
|  | Get-Command -Module PowerShellGet # Explore commands to manage PowerShell modules |
|  |  |
|  | Find-Module -Tag cloud # Find modules in the PowerShell Gallery with a "cloud" tag |
|  | Find-Module -Name ps\* # Find modules in the PowerShell Gallery whose name starts with "PS" |
|  |  |
|  | Install-Module -Name NameIT -Scope CurrentUser -Force # Install a module to your personal directory (non-admin) |
|  | Install-Module -Name NameIT -Force # Install a module to your personal directory (admin / root) |
|  | Install-Module -Name NameIT -RequiredVersion 1.9.0 # Install a specific version of a module |
|  |  |
|  | Uninstall-Module -Name NameIT # Uninstall module called "NameIT", only if it was installed via Install-Module |
|  |  |
|  | Register-PSRepository -Name <repo> -SourceLocation <uri> # Configure a private PowerShell module registry |
|  | Unregister-PSRepository -Name <repo> # Deregister a PowerShell Repository |
|  |  |
|  |  |
|  | ################################################### |
|  | # Filesystem |
|  | ################################################### |
|  |  |
|  | New-Item -Path c:\test -ItemType Directory # Create a directory |
|  | mkdir c:\test2 # Create a directory (short-hand) |
|  |  |
|  | New-Item -Path c:\test\myrecipes.txt # Create an empty file |
|  | Set-Content -Path c:\test.txt -Value '' # Create an empty file |
|  | [System.IO.File]::WriteAllText('testing.txt', '') # Create an empty file using .NET Base Class Library |
|  |  |
|  | Remove-Item -Path testing.txt # Delete a file |
|  | [System.IO.File]::Delete('testing.txt') # Delete a file using .NET Base Class Library |
|  |  |
|  | ################################################### |
|  | # Hashtables (Dictionary) |
|  | ################################################### |
|  |  |
|  | $Person = @{ |
|  | FirstName = 'Trevor' |
|  | LastName = 'Sullivan' |
|  | Likes = @( |
|  | 'Bacon', |
|  | 'Beer', |
|  | 'Software' |
|  | ) |
|  | } # Create a PowerShell HashTable |
|  |  |
|  | $Person.FirstName # Retrieve an item from a HashTable |
|  | $Person.Likes[-1] # Returns the last item in the "Likes" array, in the $Person HashTable (software) |
|  | $Person.Age = 50 # Add a new property to a HashTable |
|  |  |
|  | ################################################### |
|  | # Windows Management Instrumentation (WMI) (Windows only) |
|  | ################################################### |
|  |  |
|  | Get-CimInstance -ClassName Win32\_BIOS # Retrieve BIOS information |
|  | Get-CimInstance -ClassName Win32\_DiskDrive # Retrieve information about locally connected physical disk devices |
|  | Get-CimInstance -ClassName Win32\_PhysicalMemory # Retrieve information about install physical memory (RAM) |
|  | Get-CimInstance -ClassName Win32\_NetworkAdapter # Retrieve information about installed network adapters (physical + virtual) |
|  | Get-CimInstance -ClassName Win32\_VideoController # Retrieve information about installed graphics / video card (GPU) |
|  |  |
|  | Get-CimClass -Namespace root\cimv2 # Explore the various WMI classes available in the root\cimv2 namespace |
|  | Get-CimInstance -Namespace root -ClassName \_\_NAMESPACE # Explore the child WMI namespaces underneath the root\cimv2 namespace |
|  |  |
|  |  |
|  |  |
|  | ################################################### |
|  | # Asynchronous Event Registration |
|  | ################################################### |
|  |  |
|  | #### Register for filesystem events |
|  | $Watcher = [System.IO.FileSystemWatcher]::new('c:\tmp') |
|  | Register-ObjectEvent -InputObject $Watcher -EventName Created -Action { |
|  | Write-Host -Object 'New file created!!!' |
|  | } |
|  |  |
|  | #### Perform a task on a timer (ie. every 5000 milliseconds) |
|  | $Timer = [System.Timers.Timer]::new(5000) |
|  | Register-ObjectEvent -InputObject $Timer -EventName Elapsed -Action { |
|  | Write-Host -ForegroundColor Blue -Object 'Timer elapsed! Doing some work.' |
|  | } |
|  | $Timer.Start() |
|  |  |
|  | ################################################### |
|  | # PowerShell Drives (PSDrives) |
|  | ################################################### |
|  |  |
|  | Get-PSDrive # List all the PSDrives on the system |
|  | New-PSDrive -Name videos -PSProvider Filesystem -Root x:\data\content\videos # Create a new PSDrive that points to a filesystem location |
|  | New-PSDrive -Name h -PSProvider FileSystem -Root '\\storage\h$\data' -Persist # Create a persistent mount on a drive letter, visible in Windows Explorer |
|  | Set-Location -Path videos: # Switch into PSDrive context |
|  | Remove-PSDrive -Name xyz # Delete a PSDrive |
|  |  |
|  | ################################################### |
|  | # Data Management |
|  | ################################################### |
|  |  |
|  | Get-Process | Group-Object -Property Name # Group objects by property name |
|  | Get-Process | Sort-Object -Property Id # Sort objects by a given property name |
|  | Get-Process | Where-Object -FilterScript { $PSItem.Name -match '^c' } # Filter objects based on a property matching a value |
|  | gps | where Name -match '^c' # Abbreviated form of the previous statement |
|  |  |
|  | ################################################### |
|  | # PowerShell Classes |
|  | ################################################### |
|  |  |
|  | class Person { |
|  | [string] $FirstName # Define a class property as a string |
|  | [string] $LastName = 'Sullivan' # Define a class property with a default value |
|  | [int] $Age # Define a class property as an integer |
|  |  |
|  | Person() { # Add a default constructor (no input parameters) for a class |
|  | } |
|  |  |
|  | Person([string] $FirstName) { # Define a class constructor with a single string parameter |
|  | $this.FirstName = $FirstName |
|  | } |
|  |  |
|  | [string] FullName() { |
|  | return '{0} {1}' -f $this.FirstName, $this.LastName |
|  | } |
|  | } |
|  | $Person01 = [Person]::new() # Instantiate a new Person object. |
|  | $Person01.FirstName = 'Trevor' # Set the FirstName property on the Person object. |
|  | $Person01.FullName() # Call the FullName() method on the Person object. Returns 'Trevor Sullivan' |
|  |  |
|  |  |
|  | class Server { # Define a "Server" class, to manage remote servers. Customize this based on your needs. |
|  | [string] $Name |
|  | [System.Net.IPAddress] $IPAddress # Define a class property as an IPaddress object |
|  | [string] $SSHKey = "$HOME/.ssh/id\_rsa" # Set the path to the private key used to authenticate to the server |
|  | [string] $Username # Set the username to login to the remote server with |
|  |  |
|  | RunCommand([string] $Command) { # Define a method to call a command on the remote server, via SSH |
|  | ssh -i $this.SSHKey $this.Username@$this.Name $this.Command |
|  | } |
|  | } |
|  |  |
|  | $Server01 = [Server]::new() # Instantiate the Server class as a new object |
|  | $Server01.Name = 'webserver01.local' # Set the "name" of the remote server |
|  | $Server01.Username = 'root' # Set the username property of the "Server" object |
|  | $Server01.RunCommand("hostname") # Run a command on the remote server |
|  |  |
|  | ################################################### |
|  | # REST APIs |
|  | ################################################### |
|  |  |
|  | $Params = @{ |
|  | Uri = 'https://api.github.com/events' |
|  | Method = 'Get' |
|  | } |
|  | Invoke-RestMethod @Params # Call a REST API, using the HTTP GET method |