PowerShell Object pipeline shell

START HERE

`Get-Help <cmd> -Examples`
How do I use it?

`Where-Object <prop> -eq <val>`
Filter objects

`Get-Command *<word>*`
Find a command

`Select-Object <prop>, <prop>`
Pick properties

`<cmd> | Get-Member`
What is in the output?

`Sort-Object <prop> -Descending`
Sort objects

`Get-ChildItem`
List files

`ConvertTo-Json`
Objects to text

Get-ChildItem file objects → **Where-Object** keep some → **Sort-Object** reorder → **Select-Object** shape → **Format / Export** text at the end

Objects flow through the pipeline; $_ is the current one

CONCEPTS

Commands pass .NET objects, not text: filter and sort by property, no column parsing.

Cmdlets are named Verb-Noun. Names and parameters are case-insensitive and Tab-complete.

On Linux/macOS, ls, cat, rm and sort are not aliases: they run native tools that return text.

DISCOVER

`Get-Help <cmd>`
Show help for a command
Add -Online for the web page, -Full for everything.

`Get-Help <cmd> -Examples`
Show usage examples

`Get-Command *<word>*`
Find commands by name

`<cmd> | Get-Member`
List an object's type, properties, methods
Alias gm. The quickest way to learn what you can select.

`Update-Help`
Download help files
Help ships without full text; run once, often as admin.

PIPELINE

`Where-Object <prop> -gt <val>`
Keep objects matching a test
Alias ? or where.

`Where-Object { $_.Length -gt 1MB }`
Filter with a script block

`Select-Object <prop>, <prop>`
Keep only these properties
Alias select.

`Select-Object -First <n>`
Take the first n objects

`Select-Object -ExpandProperty <prop>`
Output the bare values

`Sort-Object <prop> -Descending`
Sort by a property

`ForEach-Object { $_.Name }`
Run a block per object
Alias %. $_ and $PSItem are the same.

PIPELINE (cont.)

`Group-Object <prop>`
Group and count by value

`Format-List *`
Show every property

FILES & PROVIDERS

`Get-ChildItem`
List items here
Aliases gci, dir; ls only on Windows.

`Get-ChildItem -Recurse -Filter <glob>`
Find files recursively

`Set-Location <path>`
Change directory
Alias cd. Push-Location / Pop-Location keep a stack.

`Get-Content <file>`
Read a file line by line
-Raw reads one string, -Tail <n> the end.

`New-Item <dir> -ItemType Directory`
Create a directory

`Copy-Item <src> <dst> -Recurse`
Copy files or folders

`Move-Item <src> <dst>`
Move or rename

`Remove-Item <path> -Recurse -Force`
Delete a tree
Add -WhatIf first to see what would go.

`Test-Path <path>`
Check if a path exists

`Select-String <pattern> <file>`
Search text, like grep
Regex by default; -SimpleMatch for literal text.

VARIABLES & TYPES

`$name = <value>`
Assign a variable

`$env:<NAME>`
Read or set an environment variable

`$a = @(1, 2, 3)`
Array; $a[-1] is the last item
+= copies the array; use a List for big loops.

`$h = @{ Key = 'val' }`
Hashtable; $h.Key or $h['Key']

`[pscustomobject]@{ A = 1; B = 2 }`
Build an object with properties

STRINGS & OPERATORS

`"Hi $name, $($obj.Prop)"`
Double quotes expand variables
Use $( ) for properties and expressions.

`'literal $text'`
Single quotes do not expand

`` `n  `t  `" ``
Escape with backtick
Backslash is a normal character.

`-eq -ne -gt -lt -ge -le`
Compare (case-insensitive)
-ceq etc. are case-sensitive. == and > do not compare.

`-like '<glob>*'`
Wildcard match

`-match '<regex>'`
Regex match; groups in $Matches

`-replace '<regex>', '<new>'`
Regex replace

DATA FORMATS & WEB

`Get-Content <file> -Raw | ConvertFrom-Json`
Parse a JSON file

`ConvertTo-Json -Depth <n>`
Serialise objects to JSON
Default depth is 2; deeper levels are cut off with a warning.

`Import-Csv <file>`
Read CSV as objects

`Invoke-RestMethod <url>`
Call an API, parse the reply
Alias irm. JSON replies become objects.

SCRIPTS & FUNCTIONS

`function <Name> { param($x) ... }`
Define a function

`param([Parameter(Mandatory)] [string]$Name)`
Required, typed parameter

`.\<script>.ps1`
Run a script
The .\ is required for the current folder.

`if (<cond>) { } elseif () { } else { }`
Conditional

`foreach ($i in $items) { }`
Loop over a collection

ERRORS

`try { } catch { $_ } finally { }`
Handle terminating errors
Most cmdlet errors are non-terminating: add -ErrorAction Stop.

`-ErrorAction Stop`
Make one command's errors catchable
Also SilentlyContinue, Ignore, Continue.

`$ErrorActionPreference = 'Stop'`
Stop on every error in scope

`$Error[0]`
Last error

`<cmd> -WhatIf`
Preview what would change

SESSION & SETUP

`$PROFILE`
Path of your startup script
Create it with New-Item -Path $PROFILE -Force.

`Set-ExecutionPolicy RemoteSigned -Scope CurrentUser`
Allow local scripts (Windows)
Not enforced on Linux/macOS. Unblock-File clears the download mark.

`Install-PSResource <module>`
Install from PowerShell Gallery
7.4+. Older: Install-Module <name> -Scope CurrentUser.

PSREADLINE KEYS

Tab / **Shift** + **Tab**
Cycle completions

Ctrl + **Space**
Menu of all completions
Windows mode (default on Windows); Linux/macOS use Emacs mode: Alt+= lists them.

Ctrl + **R**
Search history backwards

RECIPES

Biggest files below here

```
Get-ChildItem -Recurse -File |
  Sort-Object Length -Descending |
  Select-Object -First 10 FullName,
    @{ n = 'MB'; e = { [math]::Round($_.Length / 1MB, 1) } }
```