

# Windows terminal

## Tópicos de Informática para Automação

---

Mário Antunes

September 29, 2025

Universidade de Aveiro

# Welcome to the Windows Command Line!

## More Than Just a Blue Box

The **Command Line Interface (CLI)** is your direct, text-based connection to the Windows operating system.

- **Why use it?**
  - **Power & Speed:** Execute complex tasks and access settings not available in the GUI.
  - **Automation:** Script repetitive jobs with Batch or PowerShell scripts.
  - **Efficiency:** Uses minimal system resources compared to graphical tools.
  - **Industry Standard:** Essential for developers, IT professionals, and system administrators on Windows.

# The Interpreters: CMD vs. PowerShell

Windows offers two primary command-line interpreters.

- **Command Prompt (CMD):**

- The legacy interpreter for Windows, originating from MS-DOS.
- Simple, reliable for basic file operations.
- Its scripting language (Batch) is basic and less powerful.

- **PowerShell:**

- The modern, powerful, and recommended interpreter.
- Treats everything as an **object**, not just text, allowing for advanced data manipulation.
- **We will show examples for both, highlighting PowerShell's advantages.**

# The Windows Filesystem

The filesystem starts with drive letters (e.g., C:), not a single root (/).

- C:\: The root of the primary drive, where Windows is typically installed.
- C:\Users: Your personal files are here (e.g., C:\Users\Student). This is the equivalent of /home.
- C:\Windows\System32: Contains essential system programs. This is the closest equivalent to Linux's /bin.
- C:\Program Files: Default installation location for 64-bit applications.
- **The Registry:** Unlike Linux's text-based configuration in /etc, much of Windows' core configuration is stored in a hierarchical database called the Registry.

# Hidden Files & Attributes

In Windows, “hidden” is a file attribute, not just a naming convention. By default, hidden files are not shown.

## Command Prompt (dir)

```
# View only hidden files
```

```
$ dir /a:h
```

```
# View ALL files (including hidden)
```

```
$ dir /a
```

## PowerShell (ls or Get-ChildItem)

```
# View all files, including hidden and system
```

```
$ ls -Force
```

The `-Force` switch tells PowerShell to show items that would normally be hidden.

## Basic Navigation: Changing Directories (cd)

Moving around the filesystem is fundamental. The `cd` command works in both, but with a key difference.

Interpreter	Command	Description
<b>CMD</b>	<code>cd C:\Users\Student</code>	Changes the directory.
	<code>D:</code>	To change drives, type the drive letter.
<b>PowerShell</b>	<code>cd C:\Users\Student</code>	Changes the directory.
	<code>cd D:</code>	Changes the drive directly.

**Key takeaway:** PowerShell's `cd` is more intuitive as it handles path and drive changes with one command.

# Basic Navigation: Finding Your Way

Task	Command Prompt (CMD)	PowerShell
<b>Print current location</b>	cd (with no arguments)	Get-Location (alias: pwd)
<b>Go up one level</b>	cd ..	cd ..
<b>Go to user's home</b>	cd %USERPROFILE%	cd ~

# Listing Directory Contents (dir, ls)

dir and Get-ChildItem (aliased as ls) are your eyes in the terminal.

## Command Prompt (dir)

```
# Simple listing
```

```
$ dir
```

```
# Wide format, less detail
```

```
$ dir /w
```

## PowerShell (Get-ChildItem or ls)

```
# Simple listing (like Linux ls)
```

```
$ ls
```

```
# A more detailed view (like Linux ls -l)
```

```
$ ls | Format-List
```



# Creating Directories (mkdir)

Both shells use `mkdir` (or `md`), but PowerShell's is more powerful.

## Command Prompt

In CMD, you must create each level of a nested directory path one by one.

```
$ mkdir Projects  
$ mkdir Projects\TIA
```

## PowerShell

PowerShell can create the entire parent path automatically, similar to `mkdir -p` in Linux.

```
# This single command creates both 'Projects' and 'TIA'  
$ mkdir Projects\TIA
```

# Creating Files

Windows has no direct touch equivalent, so we use other methods.

## Command Prompt

Uses redirection. `echo .` creates a blank line, which is redirected to a new file.

```
# Creates an empty file
```

```
$ echo. > notes.txt
```

```
# Creates a file with content (overwrites)
```

```
$ echo My first line. > notes.txt
```

```
# Appends content to a file
```

```
$ echo My second line. >> notes.txt
```

## PowerShell

Uses the `New-Item` cmdlet for empty files and `Set-Content` for content.

```
# Creates an empty file
```

```
$ New-Item notes.txt
```

```
# Creates a file with content (overwrites)
```

```
$ Set-Content -Path notes.txt -Value "My first line."
```

```
# Appends content to a file
```

```
$ Add-Content -Path notes.txt -Value "My second line."
```

# Editing Files with Notepad

Windows does not have a built-in modern terminal editor like `nano` or `vim`.

Your primary tool for editing files from the CLI is to launch a graphical editor like **Notepad**.

```
# This works in both CMD and PowerShell  
$ notepad my_file.txt
```

This command will open `my_file.txt` in the Notepad application. If the file doesn't exist, Notepad will ask if you want to create it.

# Getting System Information (Part 1)

Task	Command Prompt (CMD)	PowerShell
<b>Current user</b>	whoami or echo %USERNAME%	whoami or \$env:USERNAME
<b>Date/Time</b>	date /t && time /t	Get-Date

## Getting System Information (Part 2)

Task	Command Prompt (CMD)	PowerShell
<b>General System Info</b>	systeminfo	Get-ComputerInfo
<b>Running Processes</b>	tasklist	Get-Process (alias: ps)

# Users & Privileges (Administrator)

Windows has two main user levels: **Standard User** and **Administrator**.

- **Administrator** is the equivalent of Linux's `root` user.
- There is no direct equivalent of `sudo`. To run a single command with elevated privileges, you must open a **new, elevated terminal**.

## How to Elevate:

1. Search for "cmd" or "powershell" in the Start Menu.
2. Right-click the icon and select **"Run as administrator."**

Any command run in this new window will have full administrative rights.

# Package Management: Winget

Modern Windows includes the **Windows Package Manager (winget)**, a command-line tool for installing software. It works in both CMD and PowerShell.

```
# Search for an application (e.g., 7zip)
```

```
$ winget search 7zip
```

```
# Install an application
```

```
$ winget install 7zip.7zip
```

```
# List installed applications
```

```
$ winget list
```

```
# Uninstall an application
```

```
$ winget uninstall 7zip.7zip
```

**Alternative:** For years, the community standard has been **Chocolatey**, which remains a very powerful alternative.



# Automation with Task Scheduler

The equivalent of cron in Windows is the **Task Scheduler**.

## Command Prompt (schtasks)

Create a task to run a script every day at 8 AM.

```
$ schtasks /create /sc daily /tn "My Task" /tr "C:\Scripts\my_script.bat" /st 08:00
```

## PowerShell (\*-ScheduledTask)

PowerShell provides a more structured way to create tasks.

```
$action = New-ScheduledTaskAction -Execute "C:\Scripts\MyScript.ps1"  
$trigger = New-ScheduledTaskTrigger -Daily -At 8am  
Register-ScheduledTask -Action $action -Trigger $trigger -TaskName "My Task"
```

# The Power of the Pipe |

The pipe sends output from one command to another. In PowerShell, this is more powerful because it sends **structured objects**, not just text.

## Command Prompt (Text Pipe)

Find the “explorer” process from the full text list.

```
$ tasklist | findstr /i "explorer"
```

## PowerShell (Object Pipe)

Get process objects, filter them, and select specific properties.

```
# Get the process object for "explorer"
$ Get-Process | Where-Object { $_.ProcessName -eq "explorer" }

# Get the process and select only its name and CPU usage
$ Get-Process "explorer" | Select-Object Name, CPU
```

# Environment Variables

Variables that store system settings. The syntax is different in each shell.

## Command Prompt

```
# View a variable using %VAR%  
$ echo %PATH%
```

```
# Set a variable for the current session  
$ set MYVAR=Hello
```

## PowerShell

```
# View a variable using $env:VAR  
$ echo $env:Path
```

```
# Set a variable for the current session  
$ $env:MYVAR="Hello"
```

**Note:** To make a variable change **permanent**, you must use the `setx` command or edit System Properties in the GUI.

# Introduction to Scripting

- **Batch Scripts (.bat, .cmd):** The traditional scripting language for CMD. Simple but clunky.
- **PowerShell Scripts (.ps1):** A modern, full-featured scripting language. Powerful and versatile.

## How to Execute Scripts

1. Save your code in a text file with the correct extension (.bat or .ps1).
2. Navigate to the directory in your terminal.
3. Run the script:
  - **CMD:** `my_script.bat`
  - **PowerShell:** `./my_script.ps1`

**PowerShell Security:** By default, running PowerShell scripts is disabled. You may need to run `Set-ExecutionPolicy RemoteSigned` in an elevated PowerShell to enable it.

# Scripting Example 1: Hello User

## Batch (hello.bat)

```
@echo off
REM Sets a variable and prints it
set USERNAME=Student
echo Hello, %USERNAME%!
```

## PowerShell (hello.ps1)

```
# Sets a variable and prints it
$Username = "Student"
Write-Host "Hello, $Username!"
```

# Scripting Example 2: If File Exists

## Batch (check\_file.bat)

```
@echo off
REM Checks if a file exists in the current directory
if exist "notes.txt" (
    echo "notes.txt was found."
) else (
    echo "notes.txt was NOT found."
)
```

## PowerShell (check\_file.ps1)

```
# Checks if a file exists in the current directory
if (Test-Path "./notes.txt") {
    Write-Host "notes.txt was found."
} else {
    Write-Host "notes.txt was NOT found."
}
```

# Scripting Example 3: Looping Through Files

## Batch (list\_files.bat)

```
@echo off
REM Lists all .txt files in the current directory
echo Found the following text files:
for %%F in (*.txt) do (
    echo - %%F
)
```

## PowerShell (list\_files.ps1)

```
# Lists all .txt files in the current directory
Write-Host "Found the following text files:"
foreach ($file in Get-ChildItem "*.txt") {
    Write-Host "- $($file.Name)"
}
```

## Final Thoughts: CMD vs. PowerShell

- **Use CMD when:** You need to run very simple, old commands or legacy batch files.
- **Use PowerShell when:** You want to perform administrative tasks, automate complex workflows, or manage Windows systems efficiently. It is the future of the Windows command line.

For any serious work, **learning PowerShell is highly recommended**. It is more powerful, consistent, and provides far better control over the Windows operating system.



Bookmark these pages for quick reference.

- **CMD Cheat Sheets:**

- [StationX CMD Cheat Sheet](#)
- [Columbia University CMD Cheatsheet](#)

- **PowerShell Cheat Sheets:**

- [Microsoft PowerShell Language Reference](#)
- [StationX PowerShell Cheat Sheet](#)