



Prompting

PowerShell

for IT Automation

Build PowerShell Scripts, CI/CD Pipelines,
Cloud Automation, Security Workflows, and
Hybrid Infrastructure Faster with
AI Prompt Engineering

A decorative graphic at the bottom of the cover featuring a wavy, undulating pattern of blue dots. The dots are arranged in a way that creates a sense of depth and movement, resembling a stylized wave or a digital signal. The pattern is more dense in the center and fades out towards the edges.

NITISH KUMAR

Copyright © 2026 Orange Education Pvt Ltd, AVA®

All rights reserved. No part of this book may be reproduced, stored in a retrieval system, or transmitted in any form or by any means, without the prior written permission of the publisher, except in the case of brief quotations embedded in critical articles or reviews.

Every effort has been made in the preparation of this book to ensure the accuracy of the information presented. However, the information contained in this book is sold without warranty, either express or implied. Neither the author nor **Orange Education Pvt Ltd** or its dealers and distributors, will be held liable for any damages caused or alleged to have been caused directly or indirectly by this book.

Orange Education Pvt Ltd has endeavored to provide trademark information about all of the companies and products mentioned in this book by the appropriate use of capital. However, **Orange Education Pvt Ltd** cannot guarantee the accuracy of this information. The use of general descriptive names, registered names, trademarks, service marks, etc. in this publication does not imply, even in the absence of a specific statement, that such names are exempt from the relevant protective laws and regulations and therefore free for general use.

First Published: May 2026

Published by: Orange Education Pvt Ltd, AVA®

Address: 9, Daryaganj, Delhi, 110002, India

275 New North Road Islington Suite 1314 London,
N1 7AA, United Kingdom

ISBN (PBK): 978-93-49887-26-8

ISBN (E-BOOK): 978-93-49887-75-6

Scan the QR code to explore our entire catalogue



www.orangeava.com

Table of Contents

1. Jumpstart PowerShell with AI Assistance

Introduction

Structure

Importance of PowerShell

AI in a Nutshell

How AI Works - from Tokens to Code

Transforming Script Writing with AI

Setting up PowerShell and AI Tools

Conclusion

2. Mastering Prompt Engineering for PowerShell

Introduction

Structure

Importance of Prompts

The Cost of Bad Prompts - Three Stories

12 Core Principles for Effective Prompts

Copy-Paste Prompt Snippets

Real-World Scenarios and Walkthroughs

Context Stacking - Conversation History

Conclusion

3. Essential PowerShell Concepts through AI-Generated Scripts

Introduction

Structure

Core Concepts Still Matter

PowerShell Data Types: An Introduction

Use AI to Play with Types

Operators and Expressions

Operators in Plain Language

Comparison Operators

String Matching Operators

Logical Operators

Arithmetic Operators

[3 Example Tricks: Learning through Contrasts](#)
[AI as a Rubber Duck for Your Logic](#)
[AI-Generated Expressions: Trust but Verify](#)
[Hands-on Micro-Exercises for Operators](#)
[Thinking in Conditions, Not Symbols](#)
[Conclusion](#)

4. Functions, Modules, and Error Handling Enhanced by AI

[Introduction](#)

[Structure](#)

[Functions and Modules Change Everything](#)

[Writing Reusable PowerShell Functions](#)

[From “chunk of code” to Named Function](#)

[Designing Cleaner Parameters with AI](#)

[Leveraging Modules: Functions Toolkit](#)

[Using AI to Sketch Module Structure](#)

[The Problem: The "Module Overhead" Tax](#)

[The AI Solution: Instant Architecture](#)

[The Human Step: Adapt and Standardize](#)

[Error Handling Basics: From Hope to try/catch](#)

[Letting AI Predict and Explain Exceptions](#)

[Reusable AI-Generated `try/catch` Patterns](#)

[Making Scripts Reliable and Maintainable](#)

[Auto-Documentation: Let AI Keep the Story Straight](#)

[Bringing It All Together: A Small AI-Assisted Toolkit](#)

[Conclusion](#)

5. AI-Enhanced PowerShell Remoting and Parallel Execution

[Introduction](#)

[Structure](#)

[Why Single-Threaded Remoting Dies at Scale](#)

[Automating Multi-Server Operations](#)

[The Wrong Tag Catastrophe](#)

[Prompt Patterns for Production-Ready Fleet Commands](#)

[Hands-on: Generate Your First AI Fleet Command](#)

[The Parallelism Trap](#)

[Vikram's AI Confession](#)

[Sequential Versus Batched Versus Full-parallel](#)
[AI as a Session Therapist](#)
[Micro-exercise: Scale a Health Check across Your Actual Lab](#)
[Security Does Not Take Weekends](#)
[Thinking in Orchestration Patterns, Not Just Commands](#)
[Conclusion](#)

6. Powerful Administrative Automation with AI Assistance

[Introduction](#)
[Structure](#)
[The Anatomy of Administrative Friction](#)
[The Stale Account Avalanche](#)
[Conversational AI as Your AD Lens](#)
[Conversation-First AD Cleanup](#)
[Policy Enforcement through Dialogue](#)
[Credential Hygiene without Syntax Nightmares](#)
[Prompt Patterns for Safe AD Operations](#)
[Generating Reports and User Management Scripts Interactively](#)
[The Over-Permission Trap](#)
[AI as Your AD Therapist](#)
[Security Boundaries in Conversational Admin](#)
[Thinking in Dialogues, Not Cmdlets](#)
[Conclusion](#)

7. Cloud Automation Made Smarter with AI

[Introduction](#)
[Structure](#)
[The Telemetry-First Azure Engineer](#)
[From Azure Portal Clicks to AI-Driven Az Scripts](#)
[Copilot for Azure as Cloud Lens](#)
[Prompt Patterns for Safe Az Operations](#)
[Managing Azure Resources with AI-Augmented Az Cmdlets](#)
[Self-Optimizing Runbooks for Compute, Storage, and Networking](#)
[Conversation-Driven Infrastructure as Code \(ARM/Bicep\)](#)
[Embedding AI in CI/CD PowerShell Workflows](#)
[Guardrails - Tags, Scopes, and FinOps Budgets](#)
[AI Agents as Cloud Housekeepers, Not Cloud Overlords](#)

Conclusion

8. Monitoring and Reporting Automation Guided by AI

Introduction

Structure

The Story-First SRE - From Noise to Narrative

Understanding the Monitoring Building Blocks

AI as Log Reader and Incident Narrator

Building Event Log Monitoring with AI-Coauthored PowerShell

Prompt Patterns for Event Log Monitoring Scripts

From KQL to AI-Guided Anomaly Detection

Designing AI-Aided Alert Policies

Teaching AI to Explain Alerts in Plain Language

Using AI to Generate HTML/CSV Monitoring Dashboards

Prompt Patterns for Dashboard and Report Automation

Automating Email and ChatOps Reporting with AI-Generated Content

Real-Time Dashboards and AI-Enhanced Workbooks

From Reactive to Predictive Monitoring with AI

Conclusion

9. Hands-On with ChatGPT for PowerShell Automation

Introduction

Structure

ChatGPT as Your PowerShell Pair-Programmer

Sending Context Securely to ChatGPT

Prompt Patterns for Script Generation and Troubleshooting

Walkthroughs of ChatGPT-Driven Script Generation

Walkthroughs of ChatGPT-Assisted Troubleshooting

Case Studies

Transforming ChatGPT Recommendations into Executable Scripts

Guardrails and Failure Modes for AI-Generated PowerShell

Conclusion

10. Boosting Developer Productivity with GitHub Copilot

Introduction

Structure

Setting up GitHub Copilot for PowerShell in VS Code

[Inline Completions: From Comments to Full Functions](#)
[Copilot Chat: Conversational PowerShell Assistance](#)
[Context Awareness: File, Workspace, and Repository Intelligence](#)
[AI-Driven Module Scaffolding and Generation](#)
[Debugging and Refactoring with Copilot](#)
[Security and Privacy Guardrails](#)
[Production Workflow Integration](#)
[Team Collaboration and Style Enforcement](#)
[Conclusion](#)

11. Leveraging Microsoft Copilot for M365 and Azure Automation

[Introduction](#)

[Structure](#)

[Using Microsoft Copilot for Enterprise-Scale Automation Scenarios](#)

[*Scenario 1: Fleet Management and Configuration Drift*](#)

[*Scenario 2: The FinOps Purge \(Cost Optimization\)*](#)

[*Scenario 3: Identity Sprawl and Cross-Tenant Governance*](#)

[*The “Guardrails” at Scale: Trust but Verify*](#)

[Streamlining Microsoft 365 Administration and Azure Governance via](#)

[AI-Guided Scripts](#)

[*The M365 Scenario: Identity Hygiene and the ‘Permanent Admin’*](#)

[*Problem*](#)

[*The Azure Scenario: Tagging, FinOps, and Shadow IT*](#)

[*The Translation Layer: From “Business Speak” to “Graph Speak”*](#)

[Hybrid AI and Scripted Solutions for Real-Time Operational Agility](#)

[*The Scenario: The “Bleeding” Update*](#)

[Conclusion](#)

12. Building AI-Augmented Automation Workflows

[Introduction](#)

[Structure](#)

[Combining PowerShell, Prompts, and Version Control](#)

[*Prompt Engineering as Code*](#)

[*The AI-Linked Commit*](#)

[*Managing the Model Drift*](#)

[*Real-World Scenario - The Monday Morning Rollback*](#)

[Creating Reusable AI-Reviewed Libraries](#)

[Managing Multi-Tool, Multi-Cloud Environments](#)
[Conclusion](#)

13. Troubleshooting and Debugging PowerShell with AI Support

[Introduction](#)

[Structure](#)

[The AI Diagnostic Framework](#)

[Interactive Debugging in VS Code](#)

[Decoding Silent Failures and Logic Bugs](#)

[The Double-Hop and Remoting Nightmares](#)

[AI-Driven Log Analysis at Scale](#)

[Real-World Scenarios](#)

[Conclusion](#)

14. AI-Driven Security and Compliance Automation

[Introduction](#)

[Structure](#)

[Implementing Regulatory Checks and Audit Reports](#)

[Automating Security Incident Response](#)

[Proactive Threat Hunting with AI](#)

[Self-Healing Infrastructure](#)

[Conclusion](#)

15. PowerShell Automation in Hybrid and Complex Environments

[Introduction](#)

[Structure](#)

[Managing Legacy and Cloud Resources with AI Assistance](#)

[AI Shell and the Agent Ecosystem](#)

[Modernizing Hybrid Infrastructure Operations](#)

[Intelligent Workload Placement and Cloud Arbitrage](#)

[Conclusion](#)

16. The Future of AI and PowerShell Automation

[Introduction](#)

[Structure](#)

[Emerging AI Trends Shaping IT Operations](#)

[Preparing for Autonomous Infrastructure Management](#)

[The Convergence - DevSecFinOps](#)
[The New Threat Landscape - Adversarial AI](#)
[The Evolution of the Community](#)
[Ethical Considerations and Human-AI Collaboration](#)
[Your Career in 2030](#)
[Conclusion](#)

[**Index**](#)

CHAPTER 1

Jumpstart PowerShell with AI Assistance

Introduction

Imagine having a task that takes eight hours to complete, and your boss expects you to do it three times a week. That is 24 hours of mind-numbing, error-prone manual work. Familiar enough? This is the exact scenario many IT professionals face as part of life. Solution? *PowerShell is the tool that turns that 8-hour nightmare into a 45-minute coffee break!*

Thus, PowerShell is essentially a super-tool for managing computers, servers, and cloud services using plain text commands. It is not just a modern and smart scripting language (auto-complete, in-built help system), but it is cross-platform as well, since it can run on Windows, Linux, and MacOS.

PowerShell has today become the "lingua franca" (universal language) for modern IT because it connects everything. I really wish PowerShell had existed when, once upon a time, I was forced to create more than 400 users within 2 hours like a typewriter, when our domain controller crashed (do not ask me why it was only the domain controller and why there was no backup).

Structure

In this chapter, we will cover the following topics:

- Importance of PowerShell
- AI in a Nutshell
- How AI Works - From Tokens to Code
- Transforming Script Writing with AI
- Setting up PowerShell and AI Tools

Importance of PowerShell

Another story, it is Monday morning, and 50 new hires are starting, but the IT team forgot to unlock their initial accounts. In the old days, an admin had to open 50 windows, find each person's name, and click "Unlock Account". That is 50

potential mistakes and an hour of work. With PowerShell, the admin just runs one simple script that reads the 50 names from a spreadsheet, and unlocks them all in less than 30 seconds.

This little story illustrates the core principle:

- **Mass Management:** PowerShell handles tasks in bulk. This is how IT teams manage **Microsoft 365** (emails, teams access) and **Azure** (cloud servers). You tell the system **what** to do, and PowerShell talks to the underlying services to get it done.
- **Consistency and Auditability:** Every time the script runs, the job is done the same way, so that it is possible to even have a transcript of what is being done. This means fewer errors, and an easy record of every action taken.

That is PowerShell, but if you are reading this book, then you already know a little about it; otherwise you would not be here, would you? Learning a new language, mastering a new way of automation, and how does these all change with the introduction of Artificial Intelligence (AI)?

Therefore, before we dive into the nuts and bolts of combining PowerShell and AI, it is essential to understand the sheer scale of the AI revolution currently sweeping across every industry. AI is not a feature; it is the new computational foundation upon which all future technologies will be built.

[AI in a Nutshell](#)

For the IT professional, AI boils down to systems that can learn and reason from data to make predictions or decisions that traditionally required human intelligence. The concepts, most relevant to an automation expert like you, are:

- **Machine Learning (ML):** The core technology that allows systems to improve performance on a specific task over time through experience (data), without being explicitly programmed.
- **Natural Language Processing (NLP):** The field that lets computers understand, interpret, and generate human language. This is what allows you to type a sentence like, "Find all large log files from yesterday and compress them," and have the computer translate it into a script.
- **Generative AI:** The cutting-edge branch (like Large Language Models, or LLMs) that can create new content—from code and configuration snippets to complex documentation—in response to a simple prompt.

Ok, so AI is the new trending buzzword in town, is it not?

Actually, not really, it is already woven into the fabric of our daily lives, changing things in very practical ways. You are probably using AI every single day without even realizing it. It is the "invisible helper" that makes your digital world work better.

Your Phone and Computer:

- **Virtual Assistants** (such as Siri, Alexa, or Google Assistant) use AI to understand your voice, answer questions, and follow commands.
- **Face ID** and your phone's camera use AI to recognize your face, and unlock your device.
- **Spam Filters** in your email use AI to instantly spot and block the junk mail.
- **Autocorrect** and predictive text on your keyboard use AI to guess what you are trying to type next.

Shopping and Entertainment:

- **Recommendations** on Netflix, Amazon, or Spotify use AI to study what you have watched or bought, and suggest things you will actually like.
- **Online Chatbots** on websites use AI to answer simple questions instantly, so that you do not have to wait for a human agent.

Getting Around:

- **Navigation Apps** (such as Google Maps or Waze) use AI to analyze real-time traffic data, and instantly help you find the fastest route.
- **Self-Driving Cars** rely entirely on AI to "see" the road, predict what other drivers will do, and make split-second decisions.

Got the point? Now, let us see how it is changing industries even.

Making Work Smarter:

- **Creative Jobs (Writers and Designers):** AI tools (often called Generative AI or Large Language Models) can draft an email, write a blog post, or create a basic image in seconds. This does not replace the human mind, but it takes away the tedious starting work, letting people focus on making the final product great!
- **Healthcare:** AI can analyze medical scans (like X-rays) faster and sometimes more accurately than a human eye, helping doctors catch diseases earlier, like finding a small tumor. It also helps speed up the development of new medicines.

- **Business:** AI is used to look at huge amounts of data to find patterns. For example, banks use it to spot fraud immediately by noticing unusual spending, and retailers use it to predict what products will be popular in the next season. Many insurance providers have even stopped costly physical inspections in favor of AI-based recommendations, whether it is even needed for the particular customer with no pattern or history of any potential fraud.
- **Repetitive Tasks:** The IT industry saw automation before, but scripted automation was rigid and rule-based. If anything changed in the environment, then scripts would need an update; if any exceptions happened that were not accounted for in the initial code, so it was a dead end. Also, it would not get better over time, rather end up in technical debt as the initial automation expert might not be there after a while. But today, **AI happens to be data-driven and adaptive**, can handle changes in real time, navigate through exceptions, and decide the best course of action. Not just that, it continuously learns from new data and feedback, becoming more accurate and efficient over time.

Hence, let us discuss some leading AI tools, and how they fit for specific types of workloads:

Tool	Primary Purpose	Key Features	Best For
ChatGPT (by OpenAI)	General-Purpose Conversational AI	Conversational chat, content creation (essays, summaries, creative writing), brainstorming, general code explanation, and debugging.	Broad-ranging tasks, creative writing, conceptual problem-solving, and general research.
GitHub Copilot (by GitHub/OpenAI)	Code Completion and Programming Assistant	Real-time code suggestions and auto-completion directly in an Integrated Development Environment (IDE). Generates functions, tests, and entire blocks of code.	Software developers accelerate coding and writing code in an IDE.
Microsoft Copilot (by Microsoft)	Productivity Assistant (Integrated)	Integrates with Microsoft 365 apps (Word, Excel, PowerPoint, Outlook, and Teams) to automate and enhance daily work tasks (for example, drafting emails,	General workplace productivity, especially for users heavily reliant on the Microsoft 365 ecosystem.

		summarizing meetings, and creating presentations).	
--	--	--	--

Table 1.1: AI Tools and Workloads Type

How AI Works - from Tokens to Code

While it feels like you are talking to a colleague who “understands” PowerShell, the underlying process is a feat of mathematical probability called **Next-Token Prediction**. To an AI, your script is not a collection of words; it is a sequence of **tokens**.

- **Tokenization - The Atomic Units of Code:** Before an AI can process your request, it breaks your text into small chunks called tokens.
 - **What is a Token?** It can be a whole word, a part of a word, or even a single punctuation mark.
 - **In PowerShell Terms:** A command like ‘Get-Service’ might be broken into three tokens: ‘Get’, ‘-’, and ‘Service’.
 - **Efficiency:** Common commands are often stored as single tokens to save “thinking” space (the context window), while complex or rare syntax is broken into smaller sub-word pieces.
- **Predicting the Next Piece of the Puzzle:** The AI does not write a whole script at once. It works like a high-speed version of “autocomplete” on your phone.
 - **The Loop:** The model looks at the tokens you provided in your prompt, and calculates which token is statistically most likely to come next.
 - **Probability Scores:** If you type ‘Get-Process |’, the model sees a high probability (perhaps 85%) that the next token should be ‘Select-Object’ or ‘Where-Object’ based on the billions of lines of code it studied during training.
 - **Autoregressive Generation:** Once it picks a token (for example, ‘Select-Object’), it adds that token to the end of the sequence, and starts the process all over again to find the **next** one. It continues this loop until it reaches a special “end” token.
- **Context and Attention:** Why does not the AI get confused in a long script? It uses a mechanism called Self-Attention.

- **Long-Range Relationships:** This allows the model to "look back" at your earlier instructions.
- **Matching Syntax:** If it opened a curly bracket ‘{’ ten lines ago, the attention mechanism reminds it that it eventually needs to predict a closing bracket ‘}’ to keep the PowerShell syntax valid.

***Tip:** This is why your prompts matter. Because the AI is just predicting the most likely “next step”, giving it a clear context (like “use PowerShell 7 syntax”) increases the probability that the tokens it predicts will be modern and accurate.*

Transforming Script Writing with AI

AI is transforming PowerShell and other script writing by making coding more intuitive, efficient, and accessible through tools like ChatGPT, GitHub Copilot, and AI Shell. These AI tools can generate scripts from natural language descriptions, help troubleshoot, optimize, and refine code, enabling faster development and smarter automation. Thus, it does not just produce scripts but provides suggestions and auto-completion as you type, not just that it can provide live recommendations, syntax corrections, and best practices in real time. Scripts are not just generated but explained line by line, helping even novices learn as they experience it over time.

Do you remember how you started with your first script in the initial days? Who helped you? What was the most instrumental in learning new ways in script, and later even debugging them? Who helped you in spotting bugs?

That one friend, one colleague, one mentor, with whom we miss coding. Do not you wish that when you get stuck in something for hours, there is someone you can ask who can help you with ideas and point out mistakes?

YES! You are getting there, AI takes the role of that friend, who is always by your side, just 24x7 now, the mentor, whose knowledge repository does not end, the smartest guy who does not take time in spotting the errors, no matter whether your code is 10 lines or 1000 lines.

Next, we would try to see that in action with simple commands and complex scenarios. Note that the outputs given are sample outputs, which we got and those might be a different code in your case, as LLMs are not supposed to be rigid but adapting and tailoring to your own pattern.

Example 1

Just open chatgpt.com, and ask *How do I use PowerShell to see the total physical memory and the currently free memory on my computer?*

You can get both the **total physical memory** and the **currently available (free) memory** using PowerShell in a few different ways.

Here are the most common and reliable methods:

🌟 Method 1: Using `Get-CimInstance` (Recommended)

```
powershell Copy code  
  
Get-CimInstance Win32_OperatingSystem | Select-Object TotalVisibleMemorySize, FreePhysicalMem
```

- `TotalVisibleMemorySize` — total physical memory in **kilobytes (KB)**.
- `FreePhysicalMemory` — currently available physical memory in **kilobytes (KB)**.

To make it more readable (e.g., show in gigabytes):

```
powershell Copy code  
  
$mem = Get-CimInstance Win32_OperatingSystem  
[PSCustomObject]@{  
    TotalMemoryGB = [math]::Round($mem.TotalVisibleMemorySize / 1MB, 2)  
    FreeMemoryGB = [math]::Round($mem.FreePhysicalMemory / 1MB, 2)  
}
```

✅ Example output:

```
diff Copy code  
  
TotalMemoryGB FreeMemoryGB  
-----  
15.89          6.72
```

Figure 1.1: One-Liner Script from ChatGPT

Example 2

Now, let us go a little further, and increase complexity. Let us ask to *write a PS script to count files, and show top file extensions.*

Here's a short PowerShell script that counts files in a directory (recursively) and shows the top file extensions by count:

```
powershell Copy code  
  
# Count files by extension and show the top 10  
Get-ChildItem -Path "C:\Your\Folder\Path" -Recurse -File |  
  Group-Object -Property Extension |  
  Sort-Object -Property Count -Descending |  
  Select-Object -First 10 Name, Count |  
  Format-Table -AutoSize
```

Explanation:

- `Get-ChildItem -Recurse -File` → Lists all files recursively.
- `Group-Object -Property Extension` → Groups files by their extension.
- `Sort-Object -Property Count -Descending` → Sorts the groups from most to least common.
- `Select-Object -First 10` → Displays the top 10 extensions.
- `Format-Table -AutoSize` → Neatly formats the output.

Figure 1.2: Complex One-Liner Script

Did it take even 5 seconds? You got a perfectly working answer within seconds yet again. Not just that, without you even asking, it explains each cmdlet, and its purpose in the scheme.

Example 3

Let us complicate things a bit further.

I want a short PS script that finds which files take up the most space on the D: drive, including hidden and system files as well. Also, ensure that during the search for files, a progress bar is also displayed.

Here's a short PowerShell script that scans your **D:** drive (including hidden and system files), finds the largest files, and shows a **progress bar** while scanning:

```
powershell Copy code

# PowerShell script to find largest files in D: drive with a progress bar

$drive = "D:\"
$files = Get-ChildItem -Path $drive -Recurse -Force -ErrorAction SilentlyContinue
$total = $files.Count
$count = 0

Write-Host "Scanning $drive for large files...\`n"

$results = foreach ($file in $files) {
    $count++
    Write-Progress -Activity "Scanning files..." `
        -Status "Processing $count of $total" `
        -PercentComplete (($count / $total) * 100)
    if (-not $file.PSIsContainer) {
        [PSCustomObject]@{
            Name = $file.FullName
            SizeMB = [math]::Round($file.Length / 1MB, 2)
        }
    }
}

# Show top 20 largest files
$results | Sort-Object SizeMB -Descending | Select-Object -First 20 | Format-Table -AutoSize
```

Figure 1.3: Multi-Line Script with Added Functionality

These three examples should give you a taste of the new age AI-assisted programming, and we have not even started yet. We have not set up anything, no complex configuration, no specific setup, just a web browser, your good old PowerShell console, and you talking in your language with a chatbot.

The fun part is, not just English, you can ask the question in Hindi or any other language, and it is supposed to give you the response as good as ever. Also, contrary to belief, even if you mistype a couple of words, LLM is supposed to understand the request as fine even then. In many cases, it may even go a step further and prompt you, if you want the output as a CSV file or HTML.

Are you enjoying the experience yet?

Setting up PowerShell and AI Tools

Step 1: Install PowerShell 7

Windows PowerShell ships with Windows versions after Windows 7, but additional steps are required to install PowerShell 7, which is the latest and cross-platform version of PowerShell. Thus, let us proceed to install the latest PowerShell 7 so that you have a modern, cross-platform shell. Verify the installation with a simple command at the end.

Windows (Choose One)

1. **Microsoft Store:** Search “PowerShell” and install PowerShell (by Microsoft). This puts `pwsh.exe` on your PATH automatically.
2. **Winget (Fastest):** Open a Windows Terminal or Windows PowerShell as Administrator, and run the following command:

```
winget install --id Microsoft.PowerShell -e
```

Close and reopen the terminal. Now, run `pwsh` to start PowerShell 7.

3. **MSI Installer (Offline/Controlled):** Download the x64 MSI from the official PowerShell releases page, and run through the wizard. Keep defaults. Start PowerShell 7 from the “start” menu.

macOS (Choose One)

1. **Homebrew (Preferred):** Install Homebrew, if needed, and then run:

```
brew install --cask powershell
```

Launch with: `pwsh`

2. **Direct Download:** Download the macOS package from the PowerShell releases page, and follow the installer prompts. Launch with: `pwsh`

Linux (Use Your Distro’s Method)

1. **Ubuntu/Debian Example:** Add Microsoft repo and install PowerShell per Microsoft’s instructions, then run: `pwsh`

Once you are done with the installation, it is time to verify the same. Open New PowerShell by typing `pwsh`, and then type `$PSversionTable` to confirm, if you have the latest PowerShell 7 installed yet.

Step 2: Install VS Code and the PowerShell Extension

Visual Studio Code (VS Code) is a free, open-source code editor developed by Microsoft that offers a lightweight yet powerful environment for coding across multiple languages and platforms. It provides features such as syntax highlighting,

intelligent code completion, debugging, integrated Git control, and an extensive marketplace for extensions that enhance functionality. Designed for speed and flexibility, VS Code supports customization through themes, keyboard shortcuts, and settings, making it a favorite among developers for web, cloud, and application development.

Install Visual Studio Code

1. Download VS Code for your OS, and install with defaults.
2. Launch VS Code after installation.

Add the PowerShell Extension

1. In VS Code, press *Ctrl+Shift+X* (macOS: *Cmd+Shift+X*) to open Extensions.
2. Search “**PowerShell**” (by Microsoft) and click Install.
3. If prompted, reload VS Code.

Quick Test

1. **File** > **New File**, save as `test.ps1`
2. Type: `Get-Process | Select-Object -First 3`
3. Press *F5* or run in the integrated terminal to confirm execution works.

Optional Tweaks

If extension activation is delayed on some setups, reload VS Code or ensure that only one PowerShell extension is enabled.

Step 3: Set up GitHub Copilot in VS Code

Now, it is time to turn on AI suggestions inside VS Code, so that comments and partial lines generate PowerShell code you can accept with Tab. This would require a GitHub account and a Copilot plan (Free plan available).

Install and Sign in

1. In VS Code, open **Extensions** and search “**GitHub Copilot**”, then click Install.
2. In the Status Bar, select the Copilot icon > Set up Copilot, then “**sign in to use Copilot**”. Authorize in the browser.
3. If you do not have access yet, the flow guides you to Copilot Free. Complete the prompts.

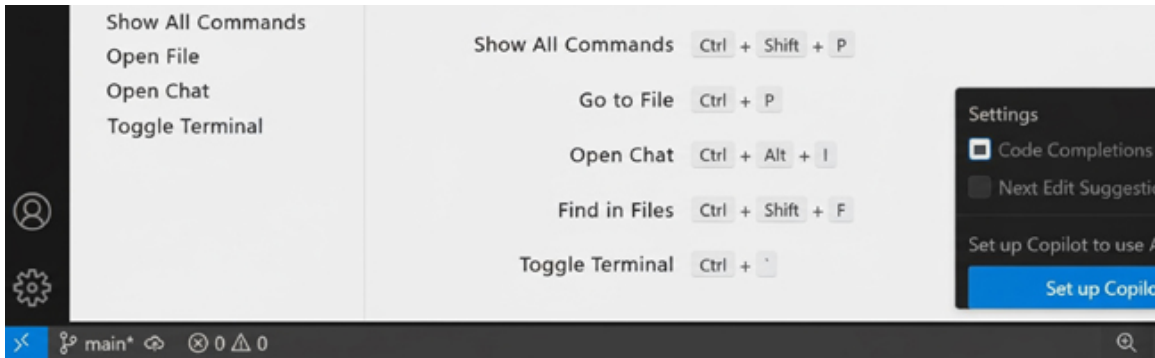


Figure 1.4: Github Copilot in VS Code

4. Confirm it works: Create a `.ps1` file, and type a helpful comment, for example:

```
# List running services with their logon accounts; export to CSV
```

A gray suggestion should appear. Press *Tab* to accept.

```
# List running services with their logon Untitled-1
1 # List running services with their logon accounts; export to CSV
2 Get-Service | Where-Object { $_.Status -eq 'Running' } | ForEach-Object {
3     $service = $_
4     $serviceWmi = Get-WmiObject -Class Win32_Service -Filter "Name='$($service.Name)"
5     [PSCustomObject]@{
6         ServiceName = $service.Name
7         DisplayName = $service.DisplayName
8         LogOnAccount = $serviceWmi.StartName
9     }
10 } | Export-Csv -Path "RunningServicesLogonAccounts.csv" -NoTypeInformation -Encoding
```

Figure 1.5: Script from a Comment Only

5. Tips for better results: Be specific in comments such as include PowerShell 7, cross-platform, WhatIf, CSV output, and any modules to use.
6. Use the Copilot panel for alternatives, if the first suggestion is not ideal.

```
# List running services with their logon Untitled-1
1 # List running services with their logon accounts; export to CSV
2 Get-Service | Where-Object { $_.Status -eq 'Running' } | ForEach-Object {
3     $service = $_
4     $serviceWmi = Get-WmiObject -Class Win32_Service -Filter "Name='$($service.Name)"
5     [PSCustomObject]@{
6         ServiceName = $service.Name
7         DisplayName = $service.DisplayName
8         LogOnAccount = $serviceWmi.StartName
9     }
10 } | Export-Csv -Path "RunningServicesLogonAccounts.csv" -NoTypeInformation -Encoding
```

Figure 1.6: Navigating Multiple Suggestions

Step 4: Install AI Shell and Chat Next to Your Prompt

Add a side-by-side chat interface in your terminal, so that you can ask questions, generate scripts, and insert code directly into your PowerShell buffer. This keeps you focused and speeds iteration.

Install AI Shell (Windows/macOS Quick Script)

1. Run the official installer script:

```
Invoke-Expression "& { $(Invoke-RestMethod  
'https://aka.ms/install-aishell.ps1') }"
```

2. This installs the **aish** executable and the AISHell PowerShell module, adding them to your PATH/module path.

Manual Install (All Platforms)

1. Download the latest AI Shell release for your OS/architecture.
2. Extract it, and add the folder to your PATH.
3. In PowerShell: `Install-PSResource -Name AISHell`
4. Restart your terminal.

Configuring AI Shell

Since the AI shell would need access to a paid model or at least credits from the free account, we would refer to generic documentation, which can be found on the link: <https://github.com/PowerShell/AISHell>.

You would need to have access to a working key to be configured in the respective json file, and configure the model of your choice as active.

Start Using AI Shell

1. In PowerShell, run: `Start-AISHell`
2. A split pane (chat) opens next to your prompt. Ask for help, for example: *“PowerShell 7 only. Create a script to list the top 10 CPU processes, and export to CSV with a timestamp.”*

```

PowerShell 7.5.0
PS C:\Users' > Start-AIShell
PS C:\Users' > Get-Process | Sort-Object CPU
PU -Descending | Select-Object -First 10 | Invoke-AIShell "what does cpu s mean?"

```

```

AI Shell
1.0.0-preview.3

Welcome to AI Shell! We're excited to have you
ur Public Preview.
Documentation is available at aka.ms/AIShell-D
e'd love to hear your
thoughts - share your feedback with us at aka.
Using the agent openai-gpt:
Active GPT: public-oai-gpt4o. A GPT instance w
expertise in PowerShell scripting and command
utilities. Use gpt-4o running in OpenAI.

Run ./help for more instructions.

@openai-gpt> what are the 10 most CPU intensi
s on windows>

To find the 10 most CPU-intensive processes
ms system, you can use PowerShell. Here's a co
will help you achieve this:

    Get-Process | Sort-Object CPU -Descending
bject -First 10

This command retrieves all running processes
em by CPU usage in descending order, and selec
10.

@openai-gpt> /code post
Code posted to the connected application.
@openai-gpt>

```

Figure 1.7: AI Shell in PowerShell Console

- When satisfied, insert the response into your terminal buffer, and review before running.

Note: *The only issue is that this is a paid option and would not work unless there is a paid model available for access. For example, using a free account, where there are no platform credits, would result in an error like the preceding one.*

```

09/11/25 C:\Users\Admin: 0.00s PS>start-AIShell
09/11/25 C:\Users\Admin: 0.00s PS>

```

```

AI Shell
1.0.0-preview.3

Using the agent openai-gpt:
Active GPT: ps-gpt4o. A GPT instance with expertise in PowerShell
scripting and command line utilities. Use gpt-4o running in OpenAI.

Run ./help for more instructions.

@openai-gpt> #1

ERROR: Agent failed to generate a response: HTTP 429 (insufficient_quota
insufficient_quota)

You exceeded your current quota, please check your plan and billing
details. For more information on this error, read the docs:
https://platform.openai.com/docs/guides/error-codes/api-errors.

```

Figure 1.8: Sample Error without Paid Subscription

As we wrap up this chapter, I hope you found yourself inspired and maybe, a little bit amazed at how the world of PowerShell has been fundamentally transformed by the rise of artificial intelligence. We have gone from marathon manual tasks and rigid scripts to dynamic, intuitive automation where your code can learn, adapt, and even saves you from those dreaded “Monday morning surprises.” If you have ever wished for a smart mentor or a tireless coding buddy, AI has now joined your toolkit, ready to troubleshoot, optimize, and teach, all at your command.

Hence, whether you are stepping into PowerShell for the first time or coaxing new tricks out of old scripts, remember: the best innovations happen when curiosity

meets creativity!

I invite you to treat every command as an experiment, every error as a lesson, and every automation challenge as a chance to stretch what is possible. The future of IT is here, and it is more collaborative, more powerful, and more fun than ever. So, grab your favorite shell, chat with your AI assistant, and continue pushing the boundaries—your next great breakthrough could be just a prompt away.

Happy scripting and happy exploring!

Conclusion

This chapter explains how PowerShell and AI combine to turn repetitive, error-prone work into fast, consistent, and auditable automation, moving from natural-language prompts to working scripts with explanations and refinements inside familiar tools such as VS Code, Copilot, and AI Shell. Along the way, you set up a complete workflow—PowerShell 7, the PowerShell extension, Copilot in-editor suggestions, and chat-driven scripting beside your terminal—while adopting a mindset that treats prompts as experiments and AI as a tireless mentor that accelerates learning and delivery across Infrastructure or other tasks.

You now have a ready environment and practical examples to iterate from one-liners to multi-step scripts with progress, filters, and exports, plus a clear path to scale reliable automation that documents itself and reduces toil. Most importantly, you have gained the confidence to pair human intent with AI assistance, turning intent into production-grade PowerShell faster and with greater clarity.

The next chapter would cover mastering Prompt Engineering for PowerShell: craft precise, context-rich prompts, apply guardrails, and use reusable templates to produce safer, more consistent scripts that align with PowerShell 7 and enterprise standards. Expect iterative patterns, evaluation checklists, and techniques that make prompts predictable, auditable, and team-ready.

You've Just Finished your Free Sample

Enjoyed the preview?

Buy: <http://www.ebooks2go.com>