A Comparative Analysis of Essential Command-Line Operations: Bash vs. Windows PowerShell

Introduction

In an era dominated by graphical user interfaces (GUIs), the command-line interface (CLI) persists not as a relic, but as an indispensable tool for power users, developers, and system administrators. The precision, scriptability, and raw power of the CLI are paramount for modern IT operations, from simple file manipulation to complex infrastructure-as-code deployments and cloud management. The ability to automate repetitive tasks and exert granular control over a system remains a core competency for any serious technical professional.

This report provides a comprehensive, expert-level analysis of two dominant command-line ecosystems: the Bash shell, which represents the heritage and philosophy of Unix-like systems, and Windows PowerShell, Microsoft's modern framework for Windows administration.

- **Bash (Bourne-Again Shell):** As the default shell for most Linux distributions and macOS, Bash is the modern torchbearer of the Unix philosophy. It is built upon a foundation of small, specialized utilities that communicate via simple text streams, enabling powerful and flexible command pipelines.[1]
- **Windows PowerShell:** Developed by Microsoft to overcome the limitations of the traditional Command Prompt, PowerShell is a task automation and configuration management framework. It is built on the.NET platform and features a powerful, object-oriented scripting language where commands (cmdlets) manipulate and pass structured objects, not just text.[2]

The central thesis of this analysis is that the fundamental architectural difference between these shells—the nature of their data pipelines—is the primary determinant of their respective toolsets, operational paradigms, and strategic use cases. While Bash excels at the rapid manipulation of unstructured text, PowerShell provides a robust, discoverable, and resilient framework for managing the complex, structured data inherent in modern systems.

A brief note on the legacy Windows Command Prompt (cmd.exe) is warranted. While still present for backward compatibility and essential for certain legacy tasks, it has been largely superseded by PowerShell for all modern administrative and automation needs.[2] This report will therefore focus on PowerShell as the primary Windows counterpart to Bash, referencing

legacy CMD tools like
icacls and mklink where they remain the standard utility for a given task.

Foundational Philosophies: Text Streams vs. Structured Objects

To comprehend the practical differences between Bash and PowerShell commands, one must first understand the divergent philosophies that underpin their design. These core principles dictate not only how individual tools are built but, more importantly, how they interact with one another.

The Unix Philosophy: A World of Text

The design of Bash and its associated utilities is a direct descendant of the Unix philosophy, which can be summarized by a few core tenets:

1. **Do one thing and do it well:** Each program should have a single, focused purpose.
2. **Write programs to work together:** The output of any program should be usable as the input to another, as-yet-unknown program.
3. **Handle text streams, because that is a universal interface:** This is the most crucial principle for this comparison. By standardizing on plain text as the medium of exchange, the Unix philosophy ensures that any tool can be combined with any other, creating a system of limitless flexibility.[1]

The primary mechanism for achieving this interoperability is the pipeline, represented by the vertical bar (|) character. In Bash, the pipeline literally connects the standard output (stdout) of the command on its left to the standard input (stdin) of the command on its right. The data passed is a simple stream of text characters.[5]

This text-centric approach has profound consequences. Its greatest strength is its universal flexibility; simple, independent tools can be composed in novel ways to solve complex problems. However, this also introduces a degree of brittleness. Scripts that rely on parsing text output can break if a future version of a command adds a new column or changes the formatting of its output. This dependency on text format necessitates a rich ecosystem of specialized parsing utilities—such as grep, sed, and awk—which have become cornerstones of shell scripting. The existence and necessity of these tools are a direct result of the pipeline's text-based nature; because the output of commands like ps or ls is unstructured text, a secondary set of tools is required to impose structure upon it for programmatic use.

The PowerShell Paradigm: A World of Objects

PowerShell was designed with a different philosophy, born from the complexities of managing the highly structured Windows operating system. Instead of text streams, PowerShell's pipeline passes full-fledged.NET objects.[2]
Commands in PowerShell, known as cmdlets, are not standalone executables that print text; they are instances of.NET classes that return structured objects. When a user pipes the output of one cmdlet to another, the entire object—complete with its properties (data) and methods (actions)—is passed down the pipeline. The next cmdlet in the chain can then directly access this structured information by name, without any need for text parsing.[6]
The consequences of this object-oriented model are the inverse of the text-based world:

- **Robustness:** Scripts are significantly more resilient to change. A script that accesses a process's name via the .ProcessName property will not break if the default display formatting of Get-Process changes in a future version of PowerShell. The data is accessed by a stable contract (the property name), not by its visual position on the screen.[7]
- **Discoverability:** The structure of any object in the pipeline can be inspected at any time using the Get-Member cmdlet. This allows a user to dynamically discover all available properties and methods, making the system highly explorable and self-documenting.
- **Verbosity:** PowerShell commands are often more verbose than their Bash counterparts. However, this verbosity typically enhances readability and clarity, as property names are explicit.

This philosophical divide explains the divergence in the command ecosystems. Bash required the evolution of powerful external text-parsing tools (grep, awk, sed) to make sense of its unstructured data streams. PowerShell, by providing structured data from the outset, instead fostered the development of generic object-filtering tools (Where-Object, Select-Object, Sort-Object) that can operate on the properties of any object passed through the pipeline.

Filesystem Mastery: Navigation, Search, and Permissions

Daily interaction with the filesystem is a fundamental task for any CLI user. The approaches taken by Bash and PowerShell for searching, managing permissions, and linking files clearly illustrate their underlying design philosophies.

Advanced File & Directory Search

Bash (find)
The find command is an exceptionally powerful and granular utility for traversing a file hierarchy. It operates by building an expression from a series of tests and actions to be

applied to each file and directory it encounters.8

- **Search by Name:** To locate all files ending with the .log extension within the /var/log directory and its subdirectories:
  Bash
  find /var/log -name "*.log"

- **Search by Size and Modification Time:** find excels at combining multiple criteria. To locate files in a user's home directory modified within the last 7 days (-mtime -7) and larger than 10 megabytes (-size +10M):
  Bash
  find /home/user -mtime -7 -size +10M

  The -mtime argument measures in 24-hour periods, while -size supports suffixes like k for kilobytes, M for megabytes, and G for gigabytes.[9]
- **Executing Commands on Results:** The -exec primary is a cornerstone of find's power, allowing an arbitrary command to be executed on each matching file. The {} placeholder is replaced by the current filename. To find and delete all temporary files (.tmp):
  Bash
  find. -name "*.tmp" -exec rm {} \;

  The command is terminated by a semicolon, which must be escaped (\;) to prevent the shell from interpreting it.[8]

PowerShell (Get-ChildItem)

PowerShell's approach uses the Get-ChildItem cmdlet (with common aliases gci, ls, or dir) to retrieve filesystem objects, which are then filtered using the pipeline.11

- **Recursive Search:** To find all .log files under the C:\Windows directory, the -Recurse parameter is used:
  PowerShell
  Get-ChildItem -Path C:\Windows -Recurse -Filter "*.log"

- **Filtering by Properties:** This is where the object-oriented nature of PowerShell becomes apparent. Instead of complex expression syntax within a single command, Get-ChildItem retrieves objects that are then filtered by Where-Object based on their properties. To find all files (-File) within the C:\Users directory that were last modified more than 30 days ago and are larger than 1 gigabyte:
  PowerShell
  Get-ChildItem -Path C:\Users -Recurse -File | Where-Object { $_.LastWriteTime -lt (Get-Date).AddDays(-30) -and $_.Length -gt 1GB }

  Here, $_.LastWriteTime and $_.Length are properties of the file objects returned by Get-ChildItem, allowing for robust and readable filtering logic.[12]

The following table provides a direct comparison for common file search tasks.

Task	Bash Command	PowerShell Command	Key Insight
Find files by name pattern (*.log)	find. -name "*.log"	Get-ChildItem -Recurse -Filter "*.log"	Both are straightforward. PowerShell's -Filter is often more performant than -Include.
Find files larger than 100MB	find. -type f -size +100M	Get-ChildItem -Recurse -File | Where-Object { $_.Length -gt 100MB }	Bash uses a built-in expression. PowerShell pipes objects to a generic filtering cmdlet.
Find files modified in the last 7 days	find. -type f -mtime -7	Get-ChildItem -Recurse -File | Where-Object { $_.LastWriteTime -gt (Get-Date).AddDays(-7) }	PowerShell's date/time logic is more explicit and leverages.NET's DateTime objects.
Find empty directories	find. -type d -empty	Get-ChildItem -Recurse -Directory | Where-Object { -not $_.GetFiles() -and -not $_.GetDirectories() }	Bash has a simple, dedicated test. PowerShell requires checking object properties/methods.
Find files by name AND size	find. -name "*.log" -size +1G	Get-ChildItem -Recurse -Filter "*.log" | Where-Object { $_.Length -gt 1GB }	Bash combines expressions. PowerShell chains filters.
Execute a command on found files	find. -name "*.tmp" -exec rm {} \;	Get-ChildItem -Recurse -Filter "*.tmp" | Remove-Item	Bash uses the -exec flag. PowerShell pipes the file objects directly to the appropriate action cmdlet (Remove-Item).

Permissions and Ownership Management

The management of file and directory permissions reveals a significant divergence in security models between the Unix-like world and Windows.

Bash (chmod, chown)

Bash operates on the classic Unix permissions model, which defines three sets of permissions for three classes of users:

- **Classes:** Owner, Group, Other (everyone else).
- **Permissions:** Read (r), Write (w), Execute (x).

The chmod (change mode) command modifies these permissions.

- **Octal Notation:** A concise method where each digit represents the permissions for Owner, Group, and Other, respectively. Read is 4, write is 2, and execute is 1. To grant the owner read/write/execute (4+2+1=7), and the group and others read/execute (4+1=5):
  Bash
  chmod 755 script.sh

- **Symbolic Notation:** A more descriptive method. To add (+) execute (x) permission for the user (owner) (u):
  Bash
  chmod u+x script.sh

The chown (change owner) command modifies the user and group ownership of a file or directory. This typically requires superuser privileges.

- **Changing User and Group:** To set the owner to www-data and the group to www-data for a web directory:
  Bash
  sudo chown www-data:www-data /var/www/html

Windows (takeown, icacls)
Windows employs a more granular model based on Access Control Lists (ACLs). An ACL is a list of Access Control Entries (ACEs), where each ACE specifies a user or group (known as a security principal) and the specific permissions they are granted or denied for that object.13
A critical distinction in the Windows security model is the primacy of ownership. Even an administrator cannot always modify the permissions of a file or folder they do not own (e.g., system files owned by the TrustedInstaller service). This necessitates a two-step process for administrative overrides: first, seize ownership, then modify the ACL. This contrasts sharply with the sudo model in Bash, where temporarily assuming the root identity grants inherent power to bypass such checks. The Windows approach forces a persistent, auditable change to the object's state (its owner) before permissions can be altered, whereas the Unix model facilitates a temporary change in the actor's state (their identity).

1. **Take Ownership (takeown):** This command allows an administrator to forcibly change the owner of a file or folder.
   - **Recursive Ownership:** To take ownership of the protected config folder and all its contents, answering "Yes" to any prompts:
     DOS
     takeown /F C:\Windows\System32\config /R /D Y

     The /R flag makes the operation recursive, and /D Y provides the default answer to prompts.[14]

2. **Modify ACLs (icacls):** The Integrity Control Access Control List (icacls) tool is used to view and modify ACLs. Its syntax is complex but powerful, allowing for fine-grained control over permissions and inheritance.[16]
    - **Granting Full Control:** To grant the user JohnDoe in the Domain domain full control (F) over the D:\Data folder, and to ensure these permissions are inherited by subfolders ((CI) - Container Inherit) and files ((OI) - Object Inherit), and to apply this recursively (/T):
      DOS
      icacls "D:\Data" /grant "Domain\JohnDoe:(OI)(CI)F" /T

      Simple permissions are denoted by letters (F for Full, M for Modify, RX for Read & Execute), while inheritance flags are enclosed in parentheses.[13]

Creating Links and Junctions

Both systems support methods for making a file or directory appear in multiple locations, but they differ in their underlying implementation.
Bash (ln)
Linux file linking is based on the concept of the inode, a data structure that stores metadata about a file, including a pointer to the actual data blocks on the disk.19

- **Hard Links:** A hard link creates a new file name (a new directory entry) that points to the *exact same inode* as the original file.
  Bash
  ln source.txt hardlink.txt

  Because both names point to the same inode, they are indistinguishable from the original file. The file's data is only removed from the disk when the link count to its inode drops to zero. Consequently, deleting the original source.txt does not affect hardlink.txt; the data remains accessible. Hard links have two main limitations: they cannot span across different filesystems, and they cannot be created for directories.[19]
- **Soft (Symbolic) Links:** A symbolic link, or symlink, is a special type of file that acts as a pointer to another file path. It has its own unique inode.
  Bash
  ln -s /data/app.log /var/log/app.log

  This creates a link named /var/log/app.log that points to the path /data/app.log. If the original file is deleted or moved, the link becomes "broken" because the path it points to is no longer valid. Symlinks are more flexible than hard links as they can link to directories and span across different filesystems.[19]

Windows (mklink)
The mklink command, a utility of the legacy Command Prompt, is the standard tool for

creating links in Windows and must be run from an elevated (Administrator) prompt.20

- **Symbolic Link (File):** The default behavior of mklink creates a file symbolic link.
  DOS
  mklink link.txt target.txt

- **Symbolic Link (Directory):** The /D switch creates a directory symbolic link. If the path contains spaces, it must be quoted.
  DOS
  mklink /D \Docs "C:\Users\User\Documents"

  This creates a symbolic link named Docs in the current directory that points to the user's Documents folder.[20]
- **Hard Link (File):** The /H switch creates a hard link, which functions similarly to its Linux counterpart, pointing to the same file data on the disk.
  DOS
  mklink /H hardlink.log original.log

- **Directory Junction:** The /J switch creates a Directory Junction, an older form of symbolic link for directories. While similar to a directory symlink, junctions are processed differently by the system and offer compatibility with older applications.[21]
  DOS
  mklink /J \Junction "C:\Path\To\Target"

Data Manipulation and Text Processing

This domain is where the philosophical divide between Bash and PowerShell is most pronounced. Bash, with its text-stream pipeline, relies on a suite of powerful, specialized tools for parsing and transforming text. PowerShell, with its object pipeline, often obviates the need for such tools by providing direct access to structured data.

Searching File Content

Bash (grep)
grep (Global Regular Expression Print) is the quintessential tool for searching plain-text data sets for lines that match a regular expression.

- **Simple Search:** To find all lines containing the string "error" in a system log:
  Bash
  grep "error" /var/log/syslog

- **Recursive and Case-Insensitive Search:** grep's power is often leveraged with its command-line flags. To search recursively (-r) through all files in the /var/log/apps/ directory for the phrase "database connection failed" while ignoring case (-i):
  Bash
  grep -Ri "database connection failed" /var/log/apps/

  This single command is a staple for administrators troubleshooting issues across numerous log files.[23]
- **Filtering and Context:** Pipelines can be used to chain grep commands for complex filtering. To exclude (-v) informational lines and then show 3 lines of context (-C 3) around any "Exception" messages:
  Bash
  grep -v "INFO" app.log | grep -C 3 "Exception"

PowerShell (Select-String)
PowerShell's counterpart to grep is Select-String (alias sls). A key difference is that Select-String does not return raw text; it returns MatchInfo objects. Each object contains properties like the filename, the line number, the matched line of text, and the pattern that was matched, making the output immediately usable for further programmatic actions.26

- **Recursive Search:** To perform the same recursive search as the grep example, one first uses Get-ChildItem to produce a stream of file objects, which are then piped to Select-String:
  PowerShell
  Get-ChildItem -Path C:\Logs -Recurse -Filter "*.log" | Select-String -Pattern "database connection failed"

  Note that by default, Select-String is case-insensitive. The -CaseSensitive switch can be added to enforce a case-sensitive match.[26]

Stream Editing and Field-Based Processing

Bash (sed and awk)
When simple searching is not enough, Bash provides sed and awk for more complex text transformation.

- **sed (Stream Editor):** Primarily used for performing search-and-replace operations. Its basic syntax is s/pattern/replacement/g, where s is the substitute command and g makes the replacement global for each line.[29]
  - **Example:** To replace every occurrence of "error" with "warning" in a log file:
    Bash
    sed 's/error/warning/g' input.log > output.log

- **awk:** A powerful, field-aware scripting language. It processes text line by line, automatically splitting each line into fields (by default, on whitespace). These fields can be referenced as $1, $2, etc..[30]
    - **Example:** To list the Process ID ($2) and command ($11) for processes whose CPU usage ($3) is greater than 5.0 percent from the output of ps aux:
      Bash
      ps aux | awk '$3 > 5.0 {print $2, $11}'

PowerShell (Object Properties and Operators)
In PowerShell, the functionality of sed and awk is often achieved not by a direct equivalent tool, but by leveraging the object pipeline and built-in operators.

- **Replacing awk:** The need to parse columns is eliminated by accessing object properties directly. The PowerShell equivalent of the awk example above is more readable and robust:
  PowerShell
  Get-Process | Where-Object { $_.CPU -gt 5.0 } | Select-Object Id, ProcessName

  Here, Get-Process outputs process objects, Where-Object filters them based on the .CPU property, and Select-Object chooses which properties to display.[7] This command is not dependent on the column order of the output.
- **Replacing sed:** Simple string substitution is handled by the -replace operator.
  PowerShell
  (Get-Content input.log) -replace 'error', 'warning' | Set-Content output.log

  Get-Content reads the file into an array of strings, the -replace operator acts on each string, and Set-Content writes the result to a new file.

This comparison reveals a fundamental trade-off: Bash offers extremely concise and powerful tools for experts who are familiar with the text formats they are manipulating. PowerShell's object-oriented approach, while more verbose, provides superior robustness, discoverability, and readability, which are highly valuable for writing maintainable, long-term automation scripts.

Sorting and Deduplication

Bash (sort and uniq)
A common text-processing pattern is to count the occurrences of unique lines in a file. In Bash, this is a two-step process because the uniq command only identifies and acts upon adjacent duplicate lines. Therefore, the input must first be sorted.33

- **Example:** The canonical one-liner to count unique lines and display them in descending order of frequency is:
  Bash

sort access.log | uniq -c | sort -nr

1. sort access.log: Sorts the lines of the file alphabetically, grouping identical lines together.
2. uniq -c: Collapses the adjacent duplicate lines into a single line, prefixed with a count (-c) of its occurrences.
3. sort -nr: Sorts the result numerically (-n) and in reverse (-r) order to show the most frequent lines first.[35]

PowerShell (Sort-Object and Get-Unique/Group-Object)
PowerShell achieves the same result through a pipeline of object-manipulation cmdlets.

- **Example:** The equivalent PowerShell pipeline is more explicit about its stages:
  PowerShell
  Get-Content access.log | Sort-Object | Get-Unique | Group-Object | Sort-Object Count -Descending

  1. Get-Content access.log: Reads the file into an array of strings.
  2. Sort-Object: Sorts the strings alphabetically.
  3. Get-Unique: Passes only the unique strings down the pipeline.
  4. Group-Object: Groups the identical items together and creates objects that have Count and Name properties.
  5. Sort-Object Count -Descending: Sorts these group objects based on their Count property.

Real-Time Log Monitoring

Bash (tail -f)
The tail command is used to display the last part of a file. Its -f (follow) option is the standard, highly efficient method for monitoring a file in real-time as new lines are appended, making it indispensable for watching log files.36

- **Example:** To watch the Apache access log for new entries:
  Bash
  tail -f /var/log/apache2/access.log

PowerShell (Get-Content -Wait)
PowerShell provides a direct equivalent with the -Wait parameter of the Get-Content cmdlet.

- **Example:** To monitor an IIS log file:
  PowerShell
  Get-Content -Path C:\inetpub\logs\LogFiles\W3SVC1\u_ex.log -Wait

  This command will display the current contents of the file and then wait, displaying new lines as they are written to the file.[38]

System Administration and Process Control

Beyond data manipulation, the CLI is the primary interface for managing the state of the operating system itself, including running processes and querying hardware information.

Process Management

Bash (ps, top, kill)
The traditional Unix toolset for process management consists of several distinct commands.

- ps: Used to get a snapshot of the current processes. The ps aux command is a common invocation to display all (a) processes for all users (u) including those without a controlling terminal (x). The output is a text table.
- top / htop: Provide an interactive, real-time view of system processes, sorted by resource usage. htop is a popular, more user-friendly alternative to the traditional top.
- kill: Sends a signal to a process, typically to terminate it. It requires the Process ID (PID) of the target. To forcefully terminate a process with PID 12345:
  Bash
  kill -9 12345

PowerShell (Get-Process, Stop-Process)
PowerShell consolidates these functions into a more integrated, object-based system.

- Get-Process (alias gps or ps): Returns a collection of Process objects, each representing a running process. This object-based output allows for powerful sorting and filtering.[39]
  - **Example:** To find the 10 processes consuming the most CPU time:
    PowerShell
    Get-Process | Sort-Object CPU -Descending | Select-Object -First 10

- Stop-Process (alias kill): Terminates one or more processes. It can target processes by name or PID and seamlessly accepts input directly from the Get-Process pipeline.[41]
  - **Example:** To find and forcefully terminate all running instances of Notepad:
    PowerShell
    Get-Process -Name "notepad" | Stop-Process -Force

This demonstrates the power of the object pipeline: Get-Process finds the process objects, and Stop-Process acts upon them without any need to manually extract and pass PIDs.[42]

Gathering System Information

Bash (A Suite of Utilities)
In the Bash environment, system information is gathered using a variety of single-purpose commands.

- df -h: Reports file system disk space usage in human-readable (-h) format.
- du -sh *: Estimates file space usage for directories (-d) in a summary (-s) human-readable (-h) format.
- uname -a: Prints all available system information, including kernel name, hostname, and kernel version.
- free -m: Displays the amount of free and used memory in the system, shown in megabytes (-m).

PowerShell (Get-PSDrive, Get-ComputerInfo)
PowerShell aims to consolidate this information into richer, more comprehensive cmdlets.

- Get-PSDrive: Provides a unified view of all "drives" available in a PowerShell session. This includes not only physical disks and mapped network shares but also other data stores exposed as drives by PowerShell providers, such as the Windows Registry (HKLM:), certificate store (Cert:), and environment variables (Env:).[43] For filesystem drives, it directly provides used and free space as object properties.
- Get-ComputerInfo: A powerful cmdlet introduced in PowerShell 5.1 that aggregates a vast amount of system information into a single, navigable object. It can retrieve details about the BIOS, operating system, processors, memory, and more, effectively replacing a multitude of legacy tools like systeminfo or wmic.[45]

"Dry Run" Operations

A "dry run" capability, which shows what a command *would* do without actually making any changes, is a critical safety feature for system administration.
Bash (Ad-hoc Implementation)
Bash itself has no universal "dry run" mechanism. This functionality is implemented on a per-command basis, if at all. For example, rsync has a --dry-run flag. For commands that lack this, such as rm, scripters must implement it manually, often by creating a wrapper function that echoes the command instead of executing it when a dry-run flag is set.47 Some Git commands also support a
--dry-run flag, such as git clean -n or git push --dry-run.[49]
PowerShell (-WhatIf)
PowerShell's design as a management framework is evident in its inclusion of "common parameters" that are available for many cmdlets. The most important of these for safety is -WhatIf. Any well-written cmdlet that makes a change to the system (a "verb" other than "Get") should support this parameter.51

- **Example:** To see which files would be deleted by a recursive Remove-Item command, without actually deleting them:

PowerShell
Remove-Item -Path C:\Temp* -Recurse -WhatIf

The command will output a list of "What if: Performing the operation 'Remove File' on target..." messages, providing a clear preview of the intended action.

The consistent availability of -WhatIf provides a system-wide, predictable safety net that is not native to the Bash ecosystem. This stems directly from PowerShell's design as a unified framework, where cmdlets are expected to adhere to a common set of behaviors, in contrast to Bash's collection of independent, heterogeneous tools.

Network Diagnostics and Configuration

Both environments provide a rich set of tools for network troubleshooting, and both have seen a clear evolution from older, less capable commands to modern, more script-friendly utilities.

IP Configuration and Details

Bash (ip addr, ifconfig)

- **Legacy (ifconfig):** For many years, ifconfig was the standard tool for configuring and viewing network interface information. It is now considered deprecated on most modern Linux systems.
- **Modern (ip):** The ip command, part of the iproute2 suite, is the modern standard. It is far more powerful and provides more detailed information in a more structured format.
    - **Example:** To display detailed IP information for the eth0 interface:
      Bash
      ip addr show eth0

PowerShell (Get-NetIPConfiguration)

- **Legacy (ipconfig):** The ipconfig command from cmd.exe is the traditional Windows tool, known for its simple text output.
- **Modern (Get-NetIPConfiguration):** This PowerShell cmdlet is the object-oriented replacement for ipconfig. It returns rich objects containing all configuration details for network interfaces.[52]
    - **Example:** To retrieve the configuration object for the network adapter with the alias "Ethernet":
      PowerShell
      Get-NetIPConfiguration -InterfaceAlias "Ethernet"

The primary advantage is the ability to directly access specific data points programmatically, such as the default gateway: (Get-NetIPConfiguration).IPv4DefaultGateway.[54] This avoids the

fragile text parsing required withipconfig.

Analyzing Network Connections

Bash (ss, netstat)

- **Legacy (netstat):** The netstat command has long been used to display network connections, routing tables, and interface statistics.
- **Modern (ss):** The ss (socket statistics) command is the modern replacement for netstat. It is significantly faster and provides more detailed information because it gets its data directly from kernel space, whereas netstat needs to parse /proc/net/tcp.[55]
    - **Example:** To show all listening (-l) TCP (-t) and UDP (-u) sockets without resolving hostnames (-n):
      Bash
      ss -tuln

PowerShell (Get-NetTCPConnection)

- **Legacy (netstat):** The netstat -an command is the traditional way to view all TCP and UDP connections and listening ports in Windows.
- **Modern (Get-NetTCPConnection):** This cmdlet provides an object-based view of TCP connections.[56] Its key advantage over
  netstat is that its output objects include the OwningProcess property (the PID), which can be easily correlated with the output of Get-Process to identify the application responsible for the connection.[55]
    - **Example:** To show established TCP connections and include the name of the owning process:
      PowerShell
      Get-NetTCPConnection -State Established | Select-Object LocalPort, RemoteAddress, @{Name="ProcessName";Expression={(Get-Process -Id $_.OwningProcess).ProcessName}}

DNS Name Resolution

Bash (dig, nslookup)

- **Legacy (nslookup):** A simple tool for querying DNS servers.
- **Modern (dig):** The dig (Domain Information Groper) utility is preferred by network administrators for its detailed, script-friendly output and advanced query options.
    - **Example:** To query Google's public DNS server (@8.8.8.8) for the A record (A) of www.google.com:
      Bash

dig A www.google.com @8.8.8.8

PowerShell (Resolve-DnsName)

- **Legacy (nslookup):** The nslookup.exe utility is available but, like its Bash counterpart, produces text output that requires parsing.
- **Modern (Resolve-DnsName):** This cmdlet is the native PowerShell tool for DNS queries. It returns structured objects, representing the DNS records themselves.[60]
    - **Example:** To perform the same query as the dig example:
      PowerShell
      Resolve-DnsName -Name www.google.com -Type A -Server 8.8.8.8

The object-based output allows for powerful command chaining. For instance, you can resolve a name and then immediately test the connection to the resulting IP address in a single pipeline.[62]
The parallel evolution in both ecosystems from older tools to more modern, script-friendly alternatives demonstrates a universal need for better automation capabilities. However, PowerShell's native object model provides a more direct and less error-prone path for integrating network data into scripts, as it eliminates the text-parsing step that is still often required in Bash, even with the improved output of modern tools like ip and ss.

Software and Environment Management

Managing the shell environment and the software installed on the system are critical administrative functions handled differently in each ecosystem.

Command-Line Package Management

Bash (A Diverse Ecosystem)
There is no single, universal package manager in the Linux world; the tool used is specific to the operating system distribution's family.

- **apt (Debian, Ubuntu):** The Advanced Package Tool is used on Debian-based systems. A typical command to update the package lists and install the htop utility without interactive prompts (-y) is:
  Bash
  sudo apt update && sudo apt install -y htop

  [63]
- **yum/dnf (Red Hat, Fedora, CentOS):** yum (Yellowdog Updater, Modified) was the long-standing package manager for Red Hat-based systems, now largely succeeded by dnf (Dandified YUM).

Bash
sudo dnf install -y htop

[65]

- **brew (macOS/Linux):** Homebrew is a popular third-party package manager that installs software in a user's home directory, avoiding the need for sudo. It has become the de facto standard on macOS and is also available for Linux.[67]
  Bash
  brew install htop

Windows (winget, Chocolatey)
For many years, Windows lacked a native command-line package manager, a void that was filled by powerful third-party tools.

- **Chocolatey:** The most mature and widely adopted third-party package manager for Windows. It has a massive community repository of packages and advanced features for managing software installations.[69]
  PowerShell
  choco install -y 7zip

- **winget:** Microsoft's official, open-source package manager, now built into modern versions of Windows. It is rapidly gaining adoption and provides a native solution for scripting software installation and management.[71]
  PowerShell
  winget install -e --id Microsoft.PowerToys

Directory Stack and Environment Variables

Bash (pushd, popd, export)

- **Directory Stack (pushd/popd):** These built-in commands manage a stack of directories. pushd /path/to/dir pushes the current directory onto the stack and changes to the new directory. popd removes the top directory from the stack and returns to it. This is highly useful in scripts for temporarily changing location and guaranteeing a return to the original directory.[73]
  Bash
  pushd /var/log # Change to /var/log, save current dir
  #... perform operations in /var/log...
  popd # Return to the original directory

- **Environment Variables (export):** In Bash, a variable is local to the current shell by default. The export command marks a variable to be passed down to the environment of

any child processes spawned from that shell.[75]
Bash
# This variable is only available in the current shell
API_KEY_LOCAL="some-value"
# This variable will be available to child processes
export API_KEY_EXPORTED="secret-value"

PowerShell (Push-Location, Pop-Location, $Env:)

- **Location Stack (Push-Location/Pop-Location):** PowerShell's cmdlets (aliased as pushd and popd) provide the same stack-based navigation. A key advantage is that they work with any PowerShell provider, not just the filesystem. One can Push-Location into a registry key, do work, and then Pop-Location back out.[77]
  PowerShell
  Push-Location HKLM:\SOFTWARE\Microsoft
  #... browse registry keys...
  Pop-Location

- **Environment Variables ($Env:):** PowerShell exposes environment variables through a PSDrive named Env:. This allows them to be managed like files.
  - **Temporary (Session-Specific):** Setting a variable this way affects only the current PowerShell session.
    PowerShell
    $Env:API_KEY = "secret-value"

  - **Persistent:** To make an environment variable persist across sessions, one must use the.NET Environment class, specifying the scope (User or Machine).
    PowerShell
    [Environment]::SetEnvironmentVariable("API_KEY", "secret-value", "User")

    This command writes the variable to the appropriate location (the registry, in this case) so it will be available in future sessions.[79]

Synthesis and Recommendations

This comparative analysis reveals two powerful and mature command-line environments, each shaped by a distinct foundational philosophy. The choice between them is not a matter of absolute superiority but of strategic alignment with the task at hand, the operating environment, and the long-term goals of automation and maintainability.

Recap: The Tale of Two Pipelines

The core difference between Bash and PowerShell remains the nature of their pipelines.

- **Bash's text-based pipeline** fosters an ecosystem of sharp, interoperable, and concise utilities (grep, awk, sed). This paradigm offers unparalleled speed and flexibility for interactive data mining and the manipulation of log files and other line-oriented text formats. Its weakness is a reliance on parsing formats that can be brittle over time.
- **PowerShell's object-based pipeline** provides a foundation of robustness, discoverability, and self-documentation. By passing structured objects instead of text, it eliminates parsing errors and enables complex automation scripts that are easier to write, read, and maintain. This makes it exceptionally well-suited for system administration, API integration, and managing the structured components of modern infrastructure.

The Rise of the Hybrid Administrator

The contemporary IT landscape is rarely homogenous. Professionals are increasingly required to operate fluently across both Windows and Linux environments. In this context, the **Windows Subsystem for Linux (WSL)** has emerged as a transformative technology. WSL provides a full, Microsoft-maintained Linux kernel that runs directly on Windows, allowing users to run native Linux distributions and their associated Bash tools seamlessly alongside PowerShell and traditional Windows applications.[2] WSL is not an emulator; it is a powerful integration that effectively dissolves the boundary between the two ecosystems on a single workstation, enabling the hybrid administrator to leverage the best tool for any given task without compromise.

Strategic Recommendations

Based on the analysis, the following strategic guidance is offered:

1. **For Intensive Text and Log File Processing:** The Bash toolchain remains the superior choice. The speed, power, and conciseness of a grep | sort | uniq -c | awk pipeline for ad-hoc analysis of unstructured text data is unmatched. On a Windows machine, these tasks should be performed within a WSL environment.
2. **For Windows System Administration and Automation:** PowerShell is the non-negotiable standard. Its deep, object-oriented integration with the Windows operating system, Active Directory, Exchange, Azure, and other Microsoft enterprise products provides a level of control and automation that is impossible to achieve with text-based tools.
3. **For Cross-Platform Scripting:** The choice is nuanced. PowerShell (version 7 and later) is cross-platform and can run on Linux and macOS. For scripts that interact heavily with structured data (like JSON or XML) or REST APIs, PowerShell can provide a consistent

experience across platforms. However, for scripts that must run on a wide array of potentially older or minimal Linux systems, sticking to POSIX-compliant shell scripting (sh) offers the highest degree of portability.

Final Insights: The Future of the CLI

The evolution of both Bash and PowerShell demonstrates a clear trajectory for the future of the command line. The trend is moving away from simple command execution and toward more powerful automation frameworks. This is evidenced by the addition of structured output formats (like JSON) to modern Linux tools and the native object-based architecture of PowerShell. The future of the CLI is one of greater structure, enhanced automation capabilities, and seamless cross-platform compatibility. Far from being a relic of the past, the command line remains a dynamic and indispensable interface for the next generation of computing challenges.

Источники

1. The Difference Between Bash and Powershell - Varonis, дата последнего обращения: августа 7, 2025, https://www.varonis.com/blog/the-difference-between-bash-and-powershell
2. Comparing CMD, PowerShell, and Windows Terminal - Spyboy blog, дата последнего обращения: августа 7, 2025, https://spyboy.blog/2025/01/04/comparing-cmd-powershell-and-windows-terminal/
3. Powershell vs Windows Subsystem for Linux? - Reddit, дата последнего обращения: августа 7, 2025, https://www.reddit.com/r/PowerShell/comments/ak84xe/powershell_vs_windows_subsystem_for_linux/
4. What is the difference between Windows Terminal, Powershell and Cmd? - Reddit, дата последнего обращения: августа 7, 2025, https://www.reddit.com/r/windows/comments/1b11hzf/what_is_the_difference_between_windows_terminal/
5. Is there a bash equivalent to a PowerShell pipeline object? - Super User, дата последнего обращения: августа 7, 2025, https://superuser.com/questions/1360096/is-there-a-bash-equivalent-to-a-powershell-pipeline-object
6. Working efficiently with the PowerShell pipeline - ScriptRunner, дата последнего обращения: августа 7, 2025, https://www.scriptrunner.com/resources/blog/working-efficiently-with-the-ps-pipeline
7. PowerShell Where-Object | Super Easy Object Filtering - MHarwood ..., дата последнего обращения: августа 7, 2025, https://mharwood.uk/powershell-where-object-super-easy-object-filtering/

8. How to Find a File in Linux | Find Command - GeeksforGeeks, дата последнего обращения: августа 7, 2025, https://www.geeksforgeeks.org/linux-unix/find-command-in-linux-with-examples/
9. Find files that match a certain size and modification time? - Unix & Linux Stack Exchange, дата последнего обращения: августа 7, 2025, https://unix.stackexchange.com/questions/258737/find-files-that-match-a-certain-size-and-modification-time
10. How to Find a File in Linux | Find Command - GeeksforGeeks, дата последнего обращения: августа 7, 2025, https://www.geeksforgeeks.org/find-command-in-linux-with-examples/
11. Get-ChildItem (Microsoft.PowerShell.Management) - Microsoft Learn, дата последнего обращения: августа 7, 2025, https://learn.microsoft.com/en-us/powershell/module/microsoft.powershell.management/get-childitem?view=powershell-7.5
12. How to use Get-ChildItem to find files - PDQ, дата последнего обращения: августа 7, 2025, https://www.pdq.com/blog/using-get-childitem-find-files/
13. Windows | How to Use the Icacls Command: Examples and Usage Guide, дата последнего обращения: августа 7, 2025, https://std.rocks/windows_icacls.html
14. Understanding how to use Icals & Takeown to make changes and reset in Windows 10, дата последнего обращения: августа 7, 2025, https://superuser.com/questions/1414910/understanding-how-to-use-icals-takeown-to-make-changes-and-reset-in-windows-10
15. Mastering icacls and takeown - .matrixpost.net, дата последнего обращения: августа 7, 2025, https://blog.matrixpost.net/mastering-icacls-and-takeown/
16. icacls | Microsoft Learn, дата последнего обращения: августа 7, 2025, https://learn.microsoft.com/en-us/windows-server/administration/windows-commands/icacls
17. Icacls | Microsoft Learn, дата последнего обращения: августа 7, 2025, https://learn.microsoft.com/en-us/previous-versions/windows/it-pro/windows-server-2012-r2-and-2012/cc753525(v=ws.11)
18. How to grant permission to users for a directory using command line in Windows? [closed], дата последнего обращения: августа 7, 2025, https://stackoverflow.com/questions/2928738/how-to-grant-permission-to-users-for-a-directory-using-command-line-in-windows
19. What Is A Hard Link In Linux : How It Differs From A Soft Link - ITU ..., дата последнего обращения: августа 7, 2025, https://www.ituonline.com/blogs/what-is-a-hard-link-in-linux/
20. Mklink Command - Computer Hope, дата последнего обращения: августа 7, 2025, https://www.computerhope.com/mklink.htm
21. Use Windows 10 mklink Command Tutorials - iSunshare, дата последнего обращения: августа 7, 2025, https://www.isunshare.com/windows-10/how-to-use-windows-10-mklink-command-tutorials.html
22. Symbolic links in windows (Example) - Coderwall, дата последнего обращения:

августа 7, 2025, https://coderwall.com/p/voy_kq/symbolic-links-in-windows
23. How to Recursively Grep all Directories and Subdirectories in Linux - GeeksforGeeks, дата последнего обращения: августа 7, 2025, https://www.geeksforgeeks.org/linux-unix/how-to-recursively-grep-all-directories-and-subdirectories-in-linux/
24. How to Grep Recursively in Bash - Stack Abuse, дата последнего обращения: августа 7, 2025, https://stackabuse.com/how-to-grep-recursively-in-bash/
25. How to Recursively Grep all Directories and Subdirectories in Linux ..., дата последнего обращения: августа 7, 2025, https://www.geeksforgeeks.org/how-to-recursively-grep-all-directories-and-subdirectories-in-linux/
26. Select-String (Microsoft.PowerShell.Utility) - Microsoft Learn, дата последнего обращения: августа 7, 2025, https://learn.microsoft.com/en-us/powershell/module/microsoft.powershell.utility/select-string?view=powershell-7.5
27. Searching for Strings in Files with PowerShell - Ironman Software, дата последнего обращения: августа 7, 2025, https://blog.ironmansoftware.com/daily-powershell/powershell-select-string/
28. Powershell equivalent of `grep -r -l` (--files-with-matches) - Super User, дата последнего обращения: августа 7, 2025, https://superuser.com/questions/742115/powershell-equivalent-of-grep-r-l-files-with-matches
29. Everything you need to know about sed substitution - learnbyexample, дата последнего обращения: августа 7, 2025, https://learnbyexample.github.io/everything-about-sed-substitution/
30. Text Processing with awk | CodeSignal Learn, дата последнего обращения: августа 7, 2025, https://codesignal.com/learn/courses/text-processing-with-bash/lessons/text-processing-with-awk
31. Efficient Text Processing in Linux: Awk, Cut, Paste - Linux Journal, дата последнего обращения: августа 7, 2025, https://www.linuxjournal.com/content/efficient-text-processing-linux-awk-cut-paste
32. Very Simple (The GNU Awk User's Guide), дата последнего обращения: августа 7, 2025, https://www.gnu.org/s/gawk/manual/html_node/Very-Simple.html
33. How to use the uniq command to process lists in Linux - Red Hat, дата последнего обращения: августа 7, 2025, https://www.redhat.com/en/blog/uniq-command-lists
34. File Manipulation: sort and uniq - FutureLearn, дата последнего обращения: августа 7, 2025, https://www.futurelearn.com/info/courses/linux-for-bioinformatics/0/steps/201955
35. Back to Basics: Sort and Uniq - Linux Journal, дата последнего обращения: августа 7, 2025, https://www.linuxjournal.com/content/back-basics-sort-and-uniq
36. The tail Command - The Linux Information Project, дата последнего обращения:

августа 7, 2025, https://www.linfo.org/tail.html
37. How To Search Logs Effectively With Log Management - Middleware, дата последнего обращения: августа 7, 2025, https://middleware.io/blog/search-logs-effectively-with-log-management/
38. Working with files and folders - PowerShell | Microsoft Learn, дата последнего обращения: августа 7, 2025, https://learn.microsoft.com/en-us/powershell/scripting/samples/working-with-files-and-folders?view=powershell-7.5
39. Get-Process (Microsoft.PowerShell.Management) - Microsoft Learn, дата последнего обращения: августа 7, 2025, https://learn.microsoft.com/en-us/powershell/module/microsoft.powershell.management/get-process?view=powershell-7.5
40. Parameters in PowerShell Get-Process - EDUCBA, дата последнего обращения: августа 7, 2025, https://www.educba.com/powershell-get-process/
41. Stop-Process (Microsoft.PowerShell.Management) - Microsoft Learn, дата последнего обращения: августа 7, 2025, https://learn.microsoft.com/en-us/powershell/module/microsoft.powershell.management/stop-process?view=powershell-7.5
42. PowerShell script to find the process and kill the process - Super User, дата последнего обращения: августа 7, 2025, https://superuser.com/questions/873601/powershell-script-to-find-the-process-and-kill-the-process
43. Get-PSDrive (Microsoft.PowerShell.Management) - Microsoft Learn, дата последнего обращения: августа 7, 2025, https://learn.microsoft.com/en-us/powershell/module/microsoft.powershell.management/get-psdrive?view=powershell-7.5
44. Managing PowerShell drives - Microsoft Learn, дата последнего обращения: августа 7, 2025, https://learn.microsoft.com/en-us/powershell/scripting/samples/managing-windows-powershell-drives?view=powershell-7.5
45. Get-ComputerInfo - PowerShell Command - PDQ, дата последнего обращения: августа 7, 2025, https://www.pdq.com/powershell/get-computerinfo/
46. Get-ComputerInfo (Microsoft.PowerShell.Management) - PowerShell ..., дата последнего обращения: августа 7, 2025, https://learn.microsoft.com/en-us/powershell/module/microsoft.powershell.management/get-computerinfo?view=powershell-7.5
47. Implementing dry-run in bash scripts - Server Fault, дата последнего обращения: августа 7, 2025, https://serverfault.com/questions/147628/implementing-dry-run-in-bash-scripts
48. Proper way to code dry run option without having to repeat myself?, дата последнего обращения: августа 7, 2025, https://unix.stackexchange.com/questions/538162/proper-way-to-code-dry-run-option-without-having-to-repeat-myself
49. Git rm - Coding Bash, дата последнего обращения: августа 7, 2025, https://codingbash.com/tutorial/how-to-use-git-rm-command-in-git-bash

50. How to Dry Run Git Commands? - GeeksforGeeks, дата последнего обращения: августа 7, 2025, https://www.geeksforgeeks.org/git/how-to-dry-run-git-commands/
51. Using PowerShell to Delete Files and Folders - Petri IT Knowledgebase, дата последнего обращения: августа 7, 2025, https://petri.com/using-powershell-to-delete-files-and-folders/
52. Get-NetIPConfiguration (NetTCPIP) | Microsoft Learn, дата последнего обращения: августа 7, 2025, https://learn.microsoft.com/en-us/powershell/module/nettcpip/get-netipconfiguration?view=windowsserver2025-ps
53. What is the PowerShell equivalent of ipconfig? | PDQ, дата последнего обращения: августа 7, 2025, https://www.pdq.com/blog/what-is-the-powershell-equivalent-of-ipconfig/
54. Windows PowerShell equivalents for common networking commands (IPCONFIG, PING, NSLOOKUP) - Jose Barreto's Blog, дата последнего обращения: августа 7, 2025, https://barreto.home.blog/2015/04/18/windows-powershell-equivalents-for-common-networking-commands-ipconfig-ping-nslookup/
55. Netstat, but Better and in PowerShell - SANS ISC, дата последнего обращения: августа 7, 2025, https://isc.sans.edu/diary/30532
56. Get-NetTCPConnection - PowerShell Command - PDQ, дата последнего обращения: августа 7, 2025, https://www.pdq.com/powershell/get-nettcpconnection/
57. How can I show the netstat command in powershell without the 0 in the Local address?, дата последнего обращения: августа 7, 2025, https://stackoverflow.com/questions/70175444/how-can-i-show-the-netstat-command-in-powershell-without-the-0-in-the-local-addr
58. Get-NetTCPConnection (NetTCPIP) | Microsoft Learn, дата последнего обращения: августа 7, 2025, https://learn.microsoft.com/en-us/powershell/module/nettcpip/get-nettcpconnection?view=windowsserver2025-ps
59. PowerShell: Get-NetTCPConnection script that also shows Username & Process Name, дата последнего обращения: августа 7, 2025, https://stackoverflow.com/questions/44509183/powershell-get-nettcpconnection-script-that-also-shows-username-process-name
60. Resolve-DnsName (DnsClient) | Microsoft Learn, дата последнего обращения: августа 7, 2025, https://learn.microsoft.com/en-us/powershell/module/dnsclient/resolve-dnsname?view=windowsserver2025-ps
61. How to Resolve DNS address using PowerShell? - Tutorialspoint, дата последнего обращения: августа 7, 2025, https://www.tutorialspoint.com/how-to-resolve-dns-address-using-powershell
62. What is the PowerShell equivalent of NSLookup? | PDQ, дата последнего обращения: августа 7, 2025, https://www.pdq.com/blog/what-is-the-powershell-equivalent-of-nslookup/

63. Install built packages - Ubuntu documentation, дата последнего обращения: августа 7, 2025, https://canonical-ubuntu-packaging-guide.readthedocs-hosted.com/en/latest/how-to/install-built-packages/
64. Install and manage packages - Ubuntu Server documentation, дата последнего обращения: августа 7, 2025, https://documentation.ubuntu.com/server/how-to/software/package-management/
65. How to use YUM to manage packages in CentOS - Hostman, дата последнего обращения: августа 7, 2025, https://hostman.com/tutorials/using-yum-in-centos/
66. 15 Linux Yum Command Examples – Install, Uninstall, Update Packages - The Geek Stuff, дата последнего обращения: августа 7, 2025, https://www.thegeekstuff.com/2011/08/yum-command-examples/
67. Homebrew — The Missing Package Manager for macOS (or Linux), дата последнего обращения: августа 7, 2025, https://brew.sh/
68. Installation - Homebrew Documentation, дата последнего обращения: августа 7, 2025, https://docs.brew.sh/Installation
69. Install-ChocolateyPackage - Chocolatey Software Docs, дата последнего обращения: августа 7, 2025, https://docs.chocolatey.org/en-us/create/functions/install-chocolateypackage/
70. Install - Chocolatey Software Docs, дата последнего обращения: августа 7, 2025, https://docs.chocolatey.org/en-us/choco/commands/install/
71. Use WinGet to install and manage applications | Microsoft Learn, дата последнего обращения: августа 7, 2025, https://learn.microsoft.com/en-us/windows/package-manager/winget/
72. Installing Windows Software from the Command Line Using WinGet - Virtualization Review, дата последнего обращения: августа 7, 2025, https://virtualizationreview.com/articles/2025/02/18/installing-windows-software-from-the-command-line-using-winget.aspx
73. pushd and popd - Wikipedia, дата последнего обращения: августа 7, 2025, https://en.wikipedia.org/wiki/Pushd_and_popd
74. Directory Navigation Using the pushd and popd Commands | Baeldung on Linux, дата последнего обращения: августа 7, 2025, https://www.baeldung.com/linux/pushd-popd
75. Defining a Bash Variable With or Without 'export' | Baeldung on Linux, дата последнего обращения: августа 7, 2025, https://www.baeldung.com/linux/bash-variables-export
76. How to set an environment variable - Autodesk, дата последнего обращения: августа 7, 2025, https://www.autodesk.com/support/technical/article/caas/sfdcarticles/sfdcarticles/How-to-set-an-environment-variable.html
77. Push-Location (Microsoft.PowerShell.Management), дата последнего обращения: августа 7, 2025, https://learn.microsoft.com/en-us/powershell/module/microsoft.powershell.management/push-location?view=powershell-7.5

78. pushd and popd in Windows PowerShell | by Pooja Nanda | CS through my eyes | Medium, дата последнего обращения: августа 7, 2025, https://medium.com/cs-through-my-eyes/pushd-and-popd-in-windows-powershell-b5fcfbbb31cd
79. How to Set environment variables using PowerShell? - Tutorialspoint, дата последнего обращения: августа 7, 2025, https://www.tutorialspoint.com/how-to-set-environment-variables-using-powershell
80. about_Environment_Variables - PowerShell - Microsoft Learn, дата последнего обращения: августа 7, 2025, https://learn.microsoft.com/en-us/powershell/module/microsoft.powershell.core/about/about_environment_variables?view=powershell-7.5