Learning basic terminal usage before diving into a version control system (VCS) like Git is essential for several reasons:

1. **Foundational Skills**: The terminal is a core tool in programming and web development. Understanding how to navigate and operate within it provides a foundation for many other tasks, including using a VCS.
2. **Efficiency**: Many version control operations are performed faster and more efficiently via the terminal. Knowing basic commands helps streamline these processes.
3. **Environment Setup**: Setting up and managing your development environment often requires terminal commands. This includes installing software, setting environment variables, and managing dependencies.
4. **Troubleshooting**: When problems arise, many solutions involve using terminal commands. Being comfortable with the terminal enables you to troubleshoot and resolve issues more effectively.
5. **Understanding Git Internals**: Many of Git’s functions and configurations are better understood through the terminal. For example, understanding file paths, permissions, and basic file operations helps in comprehending how Git tracks changes and handles repositories.
6. **Automation**: Terminal skills are crucial for automating repetitive tasks through scripts. This can significantly enhance productivity and integrate seamlessly with VCS operations.

Starting with the terminal ensures that when you move on to learning version control, you have the necessary skills to understand and utilize it effectively.

As a beginner in web development, you should start with basic terminal usage (command line). Understanding the command line will help you navigate your computer's file system, run scripts, and install necessary tools, which are essential skills for web development. Once you are comfortable with the command line, learning version control with Git will be much easier and more intuitive.

There are no strict prerequisites to learning command line usage, but having a basic understanding of the following can be helpful:

1. **Basic Computer Literacy:** Familiarity with using your operating system (Windows, macOS, or Linux), including file management and software installation.
2. **File System Structure:** Understanding how files and directories are organized on your computer.
3. **Keyboard Navigation:** Being comfortable with using the keyboard for commands rather than relying solely on the mouse.

If you have these basic skills, you should be well-prepared to start learning the command line. There are many beginner-friendly tutorials and resources available online to help you get started.

The file system structure refers to the way files and directories (folders) are organized and stored on a computer. Understanding this structure is crucial when using the command line because you will often need to navigate and manipulate files and directories. Here are the key concepts:

**1. Root Directory:**

* The root directory is the top-level directory in a file system. On Unix-like systems (Linux, macOS), it is represented by a single slash (/). On Windows, each drive has its own root directory (e.g., C:\).

**2. Directories (Folders):**

* Directories are containers that can hold files and other directories. They help organize files into a hierarchical structure.
* Example: /home/user/Documents on Unix-like systems or C:\Users\YourName\Documents on Windows.

**3. Files:**

* Files are individual pieces of data stored in the file system. They can be text files, images, programs, etc.
* Example: resume.txt, photo.jpg.

**4. Paths:**

* A path is a string that specifies the location of a file or directory within the file system.
* **Absolute Path:** Specifies the location from the root directory. Example: /home/user/Documents/file.txt (Unix-like) or C:\Users\YourName\Documents\file.txt (Windows).
* **Relative Path:** Specifies the location relative to the current directory. Example: Documents/file.txt if you're currently in the /home/user directory.

**5. Common Directories:**

* **Home Directory:** The personal directory for a user. In Unix-like systems, it is usually /home/username, and on Windows, it is C:\Users\Username.
* **Root Directory (/):** The top of the file system hierarchy in Unix-like systems.
* **System Directories:**
  + Unix-like: /bin, /etc, /usr, /var.
  + Windows: C:\Program Files, C:\Windows.

**6. Special Symbols:**

* **. (Dot):** Represents the current directory.
* **.. (Double Dot):** Represents the parent directory.

**7. File Permissions:**

* Files and directories have permissions that determine who can read, write, or execute them. Understanding basic permissions (especially on Unix-like systems) is important for managing files.

**8. Hidden Files:**

* Files and directories that are hidden from regular directory listings. On Unix-like systems, they start with a dot (e.g., .bashrc).

**Examples of Common Command Line Operations:**

* **Navigating Directories:**
  + cd /path/to/directory: Change the current directory.
  + ls: List files and directories.
* **Managing Files:**
  + cp source destination: Copy files.
  + mv source destination: Move/rename files.
  + rm filename: Remove (delete) files.

Understanding these concepts will help you effectively use the command line to navigate and manage your file system.

Not always. While PowerShell and the Windows Command Prompt (CMD) share some similarities, they have different command interpreters, syntax, and capabilities. Here’s a breakdown:

**1. Similar Commands**

Some basic commands work in both environments because PowerShell aliases many CMD commands to maintain compatibility. For example:

* dir (list files and directories)
* cd (change directory)
* copy (copy files)
* del (delete files)

These work similarly in both, but in PowerShell, they are often aliases to more advanced cmdlets.

**2. Differences in Syntax**

* **CMD commands won't always work in PowerShell** without adjustment because PowerShell is more of a scripting language with object-based processing rather than text-based.
  + Example: echo %PATH% in CMD is echo $env:PATH in PowerShell.
* **PowerShell cmdlets won't work in CMD** because CMD doesn't understand PowerShell's commands or syntax.
  + Example: Get-ChildItem (PowerShell) won't work in CMD.

**3. PowerShell-Specific Features**

PowerShell uses a more robust syntax and objects for handling commands. For instance:

* Pipes (|) in CMD pass strings, but in PowerShell, they pass objects.
* PowerShell uses cmdlets like Get-, Set-, Start-, which are far more versatile than CMD commands.

**4. Compatibility Tips**

* If you’re trying to run CMD commands in PowerShell, they might work directly (e.g., ipconfig), but environment variables and batch file syntax might need adjustments.
* You can explicitly run CMD commands in PowerShell by invoking cmd /c <command>.
* Conversely, you can run PowerShell commands in CMD using powershell -Command "<command>".

**When in doubt:**

If you're working in PowerShell, try PowerShell-native commands (Get-Help can assist). If you're in CMD, stick with commands designed for CMD.

Providing a complete list of commands for CMD (Command Prompt) and PowerShell is extensive, as both environments have numerous built-in commands. Here's how you can find them yourself along with a summarized reference:

### ****1. List of Commands for CMD****

To view all available commands in CMD:

* Open Command Prompt.
* Type help and press **Enter**. This displays a list of built-in commands.

#### **Common CMD Commands**

| **Command** | **Description** |
| --- | --- |
| cd | Change the current directory. |
| cls | Clear the screen. |
| copy | Copy files from one location to another. |
| del | Delete files. |
| dir | List files and directories. |
| echo | Display messages or turn command echoing on/off. |
| find | Search for a string in files. |
| mkdir / md | Create a new directory. |
| move | Move or rename files. |
| path | Display or set a search path for executables. |
| ping | Test network connectivity to a host. |
| rd / rmdir | Remove directories. |
| ren / rename | Rename files. |
| set | Display, set, or remove environment variables. |
| taskkill | Terminate tasks by process ID or name. |
| tasklist | Display a list of running tasks. |
| type | Display the contents of a file. |
| xcopy | Copy files and directories, including subdirectories. |
| ipconfig | Display network configuration. |

### ****2. List of Commands for PowerShell****

PowerShell has **cmdlets**, which are more powerful and versatile than CMD commands. To view all cmdlets:

* Open PowerShell.
* Type Get-Command and press **Enter**.

#### **Common PowerShell Cmdlets**

| **Cmdlet** | **Description** |
| --- | --- |
| Get-Help | Display help for cmdlets or commands. |
| Get-Command | List all available cmdlets and functions. |
| Get-Process | List currently running processes. |
| Get-Service | Display services on the system. |
| Start-Service | Start a service. |
| Stop-Service | Stop a service. |
| Set-Location (cd) | Change the current directory. |
| Get-ChildItem (dir) | List items in a directory. |
| Copy-Item (copy) | Copy files or directories. |
| Remove-Item (del) | Delete files or directories. |
| Move-Item (move) | Move or rename files/directories. |
| Get-Content (type) | View contents of a file. |
| Set-Content | Write content to a file. |
| Get-EventLog | Retrieve events from the event log. |
| Test-Connection (ping) | Test network connectivity. |
| Export-Csv | Export objects to a CSV file. |
| Import-Csv | Import data from a CSV file. |
| Invoke-WebRequest | Download data from the web. |

### ****How to Get Detailed Help****

* In CMD: Type help <command> (e.g., help dir) for details on usage.
* In PowerShell: Type Get-Help <cmdlet> (e.g., Get-Help Get-ChildItem).

### ****Exporting the Full List****

If you'd like to export all commands for further exploration:

* **CMD**: Run help > cmd\_commands.txt to save the list to a file.
* **PowerShell**: Run Get-Command > ps\_commands.txt to save the list to a file.

This will generate files with all the commands for later reference!