Basics of Shell

echo - print in shell

terminal centric applications like echo etc are shipped with the machine along with the terminal

echo $PATH : gives the list of directories, that contain the commands like echo, python etc

whenever we enter a command like echo,python, all the directories in $PATH are scanned for a matching command in them, and if it exists it runs, else says “doesn’t exist”

→ which echo : gives the path of echo in $PATH

- `date` - Displays or sets the system date and time.
- `echo` - Outputs the specified string to the terminal.
- `which` - Shows the full path of shell commands or programs that would be executed if the command is run.
- `PWD` (Present Working Directory) - Prints the full path of the current working directory.
- `CD` (Change Directory) - Changes the current working directory to a specified directory.
  - `cd /home` - Changes the directory to `/home`.
  - `cd ..` - Moves up one directory level (to the parent directory).
  - `cd ./home` - Changes to the `home` directory under the current directory.
  - `cd -` - Switches to the previous directory you were in.
- `LS` (List) - Lists files and directories in the current directory or a specified path.
  - `ls` - Lists files in the current directory.
  - `ls ..` - Lists files in the parent directory.
  - `ls /` - Lists files in the root directory.
- `.` **(Dot)** - Refers to the current directory.
- `..` **(Double Dot)** - Refers to the parent directory.

- `~` **(Tilde)** - Expands to the user's home directory.
-

1. `echo "hello" > hello.txt` - This command redirects the output of `echo` into a file named `hello.txt`. If the file doesn't exist, it creates it; if it exists, it overwrites the contents with "hello".
2. `cat hello.txt` - The `cat` command reads the contents of `hello.txt` and displays it in the terminal.
3. `cat < hello.txt` - This uses input redirection to feed the contents of `hello.txt` to the `cat` command, which then outputs it to the terminal. It's functionally the same as `cat hello.txt`.
4. `cat hello.txt > hello2.txt` - This command copies the contents of `hello.txt` to `hello2.txt`. If `hello2.txt` exists, it overwrites the file.
5. `cat hello.txt >> hello2.txt` - The `>>` operator appends the contents of `hello.txt` to the end of `hello2.txt`. If `hello2.txt` didn't exist, it would be created; if it did exist, "hello" would be added to the end of its current content.
6. `cat hello2.txt` - This command prints the contents of `hello2.txt` to the terminal. After appending, this would show "hello" twice if the previous command was used.

Explanation of Operators:

- `>` - Redirects output to a file, overwriting the file if it exists.
- `>>` - Redirects output to a file, appending to the file if it exists, instead of overwriting.

1. `|` **(Pipe Operator)** - Connects the output of one command to the input of another.
   - Example: `ls -l / | tail -n 1` - Lists the files in the root directory and then outputs only the last line.
2. `tail -n 1` - Displays the last `n` lines of a file or output. The `n 1` option means "show the last line."
3. **Redirection with** `>`, `>>`:
   - `>` - Redirects output to a file, overwriting the file.
   - `>>` - Appends the output to the end of a file without overwriting.

- Example: `ls -l / | tail -n 1 > ls.txt` - Saves the last line of the `ls -l /` output to `ls.txt`.

4. **File Permissions and `sudo`**:
   - `sudo` - Runs a command with superuser (root) privileges.
   - Example: `sudo su` - Opens a root shell, allowing you to execute commands as the root user.
   - `sudo tee` - A combination of `sudo` and `tee` to write output to a file as root without needing to enter a root shell.
   - Example: `echo 1060 | sudo tee brightness` - Writes `1060` to the `brightness` file with root privileges.
5. `tee` - Takes input, writes it to a file, and also outputs it to the terminal.
   - Example: `echo 1060 | sudo tee brightness` - Outputs `1060` to the terminal and writes it to `brightness`.
6. `find` - Searches for files and directories that match a specified pattern.
   - Example: `find . -name "brightness"` - Searches for files named "brightness" in the current directory.
7. `echo 1 > /sys/class/leds/scrolllock/brightness` - This writes `1` to the `brightness` file for the scroll lock LED, turning it on. This is an example of how you can control hardware settings by writing directly to system files.

Concepts:

- **Shell Behavior and Redirection**: The shell handles redirection, not the commands themselves. For instance, when using `sudo`, the redirection is processed by the shell running under the user's permissions, not the elevated `sudo` command.
- **Root vs. Non-Root User**: Commands executed as root can modify critical system files, while non-root users may face permission issues when attempting the same.
- `scroll lock` **LED**: Demonstrates the concept of controlling hardware through the file system by directly writing to special files managed by the kernel.
- **Opening Files from the Terminal**:
  - `xdg-open`: A command typically used on Linux to open a file with the default application associated with its file type. For example, running

`xdg-open file.html` will open the file in your default web browser.

- `open` **(macOS)**: The macOS equivalent of `xdg-open`. You can use it in the same way to open files from the terminal.
- **Windows Alternative**: The method to open files directly from the terminal in Windows is not specified, but typically it involves commands like `start` or `explorer`.

- **Minimizing GUI Dependency**:
  - The idea is to reduce reliance on point-and-click interfaces (like Finder on macOS or File Explorer on Windows) by using terminal commands to find and open files, improving efficiency and integrating with automation processes.