* Commands to show how to interact with the filesystem using basic shell commands.

Interacting with the filesystem using basic shell commands is fundamental to navigating, creating, modifying, and managing files and directories. Here are some essential shell commands for interacting with the filesystem:

**Navigating and Listing Files/Directories**

1. **pwd**: Print the current working directory.

bash

Copy code

pwd

1. **ls**: List directory contents.

bash

Copy code

ls

ls -l # Detailed list view

ls -a # Show hidden files

ls -lh # Human-readable sizes

1. **cd**: Change directory.

bash

Copy code

cd /path/to/directory

cd .. # Move up one directory

cd ~ # Go to the home directory

**Creating and Removing Files/Directories**

1. **mkdir**: Create directories.

bash

Copy code

mkdir directory\_name

mkdir -p path/to/nested/directory # Create nested directories

1. **touch**: Create empty files or update file timestamps.

bash

Copy code

touch filename

1. **rm**: Remove files or directories.

bash

Copy code

rm filename

rm -r directory\_name # Remove directory recursively

**Viewing and Editing Files**

1. **cat**: Display file content.

bash

Copy code

cat filename

1. **less or more**: View file contents one screen at a time.

bash

Copy code

less filename

more filename

1. **head and tail**: Display the beginning or end of a file.

bash

Copy code

head filename # Display first few lines

tail filename # Display last few lines

1. **vi or vim**: Text editor for creating or editing files.

bash

Copy code

vi filename

**Moving, Copying, and Renaming Files/Directories**

1. **mv**: Move or rename files/directories.

bash

Copy code

mv file1 file2 # Rename file1 to file2

mv file1 directory # Move file1 to directory

1. **cp**: Copy files/directories.

bash

Copy code

cp file1 file2 # Copy file1 to file2

cp -r directory destination\_directory # Copy directory recursively

**Permissions and Ownership**

1. **chmod**: Change file permissions.

bash

Copy code

chmod +x script.sh # Add execute permission

chmod 644 file.txt # Set permissions (owner: read/write, group: read, others: read)

1. **chown**: Change file ownership.

bash

Copy code

chown user:group filename # Change owner and group of filename

**Searching for Files**

1. **find**: Search for files in a directory hierarchy.

bash

Copy code

find /path/to/search -name "\*.txt" # Find all .txt files in /path/to/search

1. **grep**: Search for patterns in files.

bash

Copy code

grep "pattern" filename

**Miscellaneous**

1. **du**: Disk usage of files and directories.

bash

Copy code

du -h # Human-readable sizes

1. **df**: Disk free space.

bash

Copy code

df -h # Human-readable sizes

1. **ln**: Create links between files.

bash

Copy code

ln -s /path/to/file link\_name # Create symbolic link

These basic shell commands allow you to effectively navigate, manipulate, and manage files and directories within the filesystem. Combine these commands with shell scripting for more complex tasks and automation. Adjust command options and parameters as needed based on your specific requirements and environment.

* Show how to use the vim text editor to edit text at the terminal.

Using the Vim text editor at the terminal allows you to create, edit, and modify text files directly from the command line interface. Here’s a basic guide on how to use Vim:

**Opening or Creating a File**

To open or create a file with Vim, use the following command:

bash

Copy code

vim filename

Replace filename with the name of the file you want to open or create. If the file doesn't exist, Vim will create a new file with that name.

**Vim Modes**

Vim operates in different modes:

1. **Normal Mode**: This is the default mode where you can navigate the file and issue commands.
2. **Insert Mode**: Allows you to insert or edit text within the file.
3. **Command-Line Mode**: Used for executing commands like saving files or quitting Vim.

**Basic Vim Commands**

Once you open a file in Vim, you're initially in **Normal Mode**. Here are some essential commands to get started:

* **Switching to Insert Mode**:
  + Press i to enter **Insert Mode**. You'll see -- INSERT -- at the bottom of the screen.
  + Now you can start typing to insert or edit text.
* **Saving and Exiting**:
  + **Command-Line Mode**:
    - Press Esc to ensure you are in **Normal Mode**.
    - Type :w and press Enter to save changes (w stands for write).
    - Type :wq and press Enter to save and quit Vim (wq stands for write and quit).
    - Type :q! and press Enter to quit without saving changes (q! stands for quit and discard changes).
* **Navigating in Normal Mode**:
  + Use arrow keys or h, j, k, l keys (h for left, j for down, k for up, l for right).
  + gg moves to the beginning of the file.
  + G moves to the end of the file.
  + :line\_number moves to a specific line number (e.g., :10 moves to line 10).
* **Editing Text**:
  + Delete a character: Press x.
  + Delete a line: Press dd.
  + Undo: Press u.
  + Redo: Press Ctrl + r.
* **Searching**:
  + Type / followed by your search term and press Enter to search forward.
  + Type ? followed by your search term and press Enter to search backward.
  + Use n to move to the next search result and N to move to the previous search result.
* **Copying and Pasting**:
  + Position the cursor where you want to start copying.
  + Press v to enter **Visual Mode** (for character-wise visual selection) or V (for line-wise visual selection).
  + Move the cursor to select text.
  + Press y to copy (yank) the selected text.
  + Position the cursor where you want to paste.
  + Press p to paste the copied text after the cursor or P to paste before the cursor.

**Additional Tips**

* To learn more about Vim, you can use vimtutor in the terminal, which provides a basic interactive tutorial on Vim commands and usage.
* Vim has extensive capabilities beyond these basics. Exploring and practicing with Vim will help you become proficient over time.

Using Vim efficiently requires practice, but mastering it can greatly enhance your productivity, especially when working in a terminal environment where graphical editors may not be available.

Top of Form

Bottom of Form

* Commands to show how to redirect the flow of data between STDIN/STDOUT/STDERR

Redirecting the flow of data between STDIN (standard input), STDOUT (standard output), and STDERR (standard error) is crucial for manipulating and managing data streams in Unix-like operating systems. Here are the basic commands and operators used for redirection:

**1. Redirect STDOUT to a File**

* **>**: Redirects STDOUT to a file, overwriting the file if it exists.

bash

Copy code

command > output.txt

* **>>**: Redirects STDOUT to a file, appending to the file if it exists.

bash

Copy code

command >> output.txt

**2. Redirect STDERR to a File**

* **2>**: Redirects STDERR to a file, overwriting the file if it exists.

bash

Copy code

command 2> error.txt

* **2>>**: Redirects STDERR to a file, appending to the file if it exists.

bash

Copy code

command 2>> error.txt

**3. Redirect STDOUT and STDERR to Different Files**

* **> and 2>**: Redirects STDOUT to one file and STDERR to another file.

bash

Copy code

command > output.txt 2> error.txt

**4. Redirect STDERR to STDOUT**

* **2>&1**: Redirects STDERR to the same location as STDOUT.

bash

Copy code

command 2>&1

**5. Redirect STDOUT to STDERR**

* **1>&2**: Redirects STDOUT to STDERR.

bash

Copy code

command 1>&2

**6. Redirecting Both STDOUT and STDERR to a File**

* **&> or >&**: Redirects both STDOUT and STDERR to a file, overwriting the file if it exists.

bash

Copy code

command &> output.txt

or

bash

Copy code

command >& output.txt

* **&>>**: Redirects both STDOUT and STDERR to a file, appending to the file if it exists.

bash

Copy code

command &>> output.txt

**7. Discard STDOUT or STDERR**

* **>/dev/null**: Discards the STDOUT output.

bash

Copy code

command > /dev/null

* **2>/dev/null**: Discards the STDERR output.

bash

Copy code

command 2> /dev/null

* **&>/dev/null or &>>/dev/null**: Discards both STDOUT and STDERR output.

bash

Copy code

command &>/dev/null

or

bash

Copy code

command &>>/dev/null

**Example Usage**

**Redirect STDOUT to a File:**

bash

Copy code

ls > directory\_contents.txt

**Redirect STDERR to a File:**

bash

Copy code

cat non\_existent\_file.txt 2> error.log

**Redirect STDOUT and STDERR to Different Files:**

bash

Copy code

ls /path/to/directory > directory\_contents.txt 2> ls\_errors.log

**Redirect STDERR to STDOUT:**

bash

Copy code

cat non\_existent\_file.txt 2>&1

**Redirect STDOUT to STDERR:**

bash

Copy code

echo "Error message" 1>&2

**Redirect Both STDOUT and STDERR to a File:**

bash

Copy code

ls /path/to/directory &> ls\_output\_and\_errors.txt

**Discard STDOUT:**

bash

Copy code

echo "Hello" > /dev/null

**Discard STDERR:**

bash

Copy code

cat non\_existent\_file.txt 2> /dev/null

**Discard Both STDOUT and STDERR:**

bash

Copy code

command &> /dev/null

These redirection techniques are essential for managing command output and error handling in scripts and interactive terminal sessions. They provide flexibility in how you capture, discard, and manage data streams within the Unix shell environment.

Top of Form

Bottom of Form

* Create a shell script that creates 6 file each of types .png, .mp3, and .mp4. Move these to Image, Songs, and Movies folders respectively.

To create a shell script that creates files of specific types (.png, .mp3, .mp4) and organizes them into corresponding directories (Image, Songs, Movies), you can use the following script:

bash

Copy code

#!/bin/bash

# Create directories if they don't exist

mkdir -p Image

mkdir -p Songs

mkdir -p Movies

# Create 6 files of each type (.png, .mp3, .mp4)

for i in {1..6}; do

touch "file\_${i}.png"

touch "file\_${i}.mp3"

touch "file\_${i}.mp4"

done

# Move files to their respective directories

mv \*.png Image/

mv \*.mp3 Songs/

mv \*.mp4 Movies/

echo "Files created and organized successfully."

**Explanation:**

1. **Shebang (#!/bin/bash)**: Indicates that the script should be executed using the Bash shell.
2. **Create Directories (mkdir -p)**:
   * mkdir -p Image: Creates the Image directory if it doesn't exist.
   * mkdir -p Songs: Creates the Songs directory if it doesn't exist.
   * mkdir -p Movies: Creates the Movies directory if it doesn't exist.
3. **Create Files (for loop)**:
   * Uses a for loop to create 6 files of each type (.png, .mp3, .mp4) using the touch command.
4. **Move Files (mv command)**:
   * Moves all .png files to the Image directory.
   * Moves all .mp3 files to the Songs directory.
   * Moves all .mp4 files to the Movies directory.
5. **Output Message**: Prints a message indicating that files have been created and organized successfully.

**Usage:**

* Save the script into a file, for example, organize\_files.sh.
* Make the script executable:

bash

Copy code

chmod +x organize\_files.sh

* Run the script:

bash

Copy code

./organize\_files.sh

This script will generate 6 files of each type (.png, .mp3, .mp4) in the current directory and then move them into their respective folders (Image, Songs, Movies). Adjust file names or paths as needed based on your specific requirements.

Top of Form

Bottom of Form