Activity Guide: Building a Raspberry Pi System Monitor

**Objective:** Create a bash script on a Raspberry Pi that functions as a system monitor, incorporating elements of operating systems, file systems, software, applications, web browsers, software licensing, and basic Linux commands.

### **Activity 1: Project Setup and Planning**

1. **Project Scope and Objectives:**
   * Clearly define the scope and objectives of the project.
   * Identify key features and functionalities of the system monitor script.
2. **Linux Commands and Libraries:**
   * List the Linux commands and libraries you plan to use in the script.
   * Research and ensure compatibility with the Raspberry Pi environment.
3. **Design the Script:**
   * Sketch a high-level overview of the script's architecture and flow.
   * Plan the menu-driven interface and user interaction.

### **Activity 2: Script Implementation**

1. **Script Structure:**
   * Create a new bash script file (e.g., system\_monitor.sh).
   * Define the overall structure, including functions and main execution flow.
2. **Operating System Information:**
   * Use Linux commands (lsb\_release, uname, etc.) to gather and display information about the operating system.
   * Implement a function to present this information in a user-friendly format.
3. **File System Details:**
   * Utilize commands (df, ls, etc.) to extract details about the file system.
   * Design a section in the script to display storage capacity, file types, and directory structures.
4. **Software and Applications:**
   * Identify installed software using commands (dpkg, apt, etc.).
   * Create a menu option to display information about software versions and dependencies.
   * Implement functionality to launch basic applications.

**1. Create the Script File:**

* Open a text editor on your Raspberry Pi (e.g., nano system\_monitor.sh).

**2. Define Script Structure:**

* Start with the shebang line to indicate that this is a bash script.
* Define any global variables and constants.

#!/bin/bash

# Global Variables

SCRIPT\_VERSION="1.0"

**3. Implement Operating System Information:**

* Create a function to gather and display information about the operating system.

get\_os\_info() {

echo "### Operating System Information ###"

echo "Distributor: $(lsb\_release -d -s)"

echo "Kernel: $(uname -r)"

echo "Arch: $(uname -m)"

echo "####################################"

}

**4. Implement File System Details:**

* Create a function to extract and display details about the file system.

get\_file\_system\_info() {

echo "### File System Details ###"

echo "Disk Usage:"

df -h

echo "----------------------------"

echo "Directory Listing:"

ls -l

echo "############################"

}

**5. Main Execution Flow:**

* In the main part of the script, call the functions and provide a menu-driven interface.

main() {

clear # Clear the terminal screen

echo "Raspberry Pi System Monitor (v$SCRIPT\_VERSION)"

echo "--------------------------------------------"

PS3="Select an option: "

options=("Get OS Information" "Get File System Details" "Quit")

select opt in "${options[@]}"; do

case $opt in

"Get OS Information")

get\_os\_info

;;

"Get File System Details")

get\_file\_system\_info

;;

"Quit")

echo "Exiting the System Monitor. Goodbye!"

exit 0

;;

\*)

echo "Invalid option. Please try again."

;;

esac

done

}

# Call the main function to start the script

main

**6. Save and Run:**

* Save the script file (Ctrl + O in nano) and exit the text editor (Ctrl + X in nano).
* Make the script executable: chmod +x system\_monitor.sh.
* Run the script: ./system\_monitor.sh.

### **Activity 3: Additional Features**

1. **Web Browser Integration:**
   * Incorporate functionality to launch a web browser using commands (chromium-browser, etc.).
   * Display basic information about the default web browser.
2. **Software Licensing:**
   * Integrate commands (licensecheck, etc.) to extract information about software licensing.
   * Present details on open-source licenses for relevant software.
3. **Real-time Monitoring (Bonus):**
   * Use the psutil library (in Python) to implement real-time monitoring of system resources.
   * Update the script to display CPU, memory, and disk usage dynamically.

**1. Implement Web Browser Integration:**

* Create a function to launch the default web browser on the Raspberry Pi.

launch\_web\_browser() {

echo "Launching the default web browser..."

chromium-browser # Change this to the actual command for your default web browser

}

* Update the main execution flow to include an option for launching the web browser.

options=("Get OS Information" "Get File System Details" "Launch Web Browser" "Quit")

case $opt in

"Launch Web Browser")

launch\_web\_browser

;;

# ... (other cases remain unchanged)

**2. Integrate Software Licensing Information:**

* Create a function to extract and display information about software licensing.

get\_software\_licensing\_info() {

echo "### Software Licensing Information ###"

# Use appropriate commands to extract licensing information

licensecheck --recursive .

echo "######################################"

}

* Add an option for displaying software licensing information in the main execution flow.

options=("Get OS Information" "Get File System Details" "Launch Web Browser" "Software Licensing Info" "Quit")

case $opt in

"Software Licensing Info")

get\_software\_licensing\_info

;;

# ... (other cases remain unchanged)

**3. Save and Run:**

* Save the script file (Ctrl + O in nano) and exit the text editor (Ctrl + X in nano).
* Make the script executable: chmod +x system\_monitor.sh.
* Run the script: ./system\_monitor.sh.

### **Activity 4: Testing and Feedback**

1. **Initial Testing:**
   * Execute the script on your Raspberry Pi to ensure basic functionality.
   * Identify and fix any issues that arise during initial testing.
2. **User Feedback:**
   * Share the script with potential users or testers.
   * Collect feedback on usability and any identified improvements.
3. **Iterative Improvement:**
   * Make necessary adjustments to the script based on user feedback.
   * Test the updated version to ensure improvements and maintain functionality.

### **Activity 5: Documentation**

1. **README File:**
   * Create a README file with information on how to run the script, dependencies, and setup requirements.
2. **Function Documentation:**
   * Document each function and section of the script.
   * Include comments explaining the purpose and usage of code segments.
3. **User Guide:**
   * Write a user guide that explains menu options, interpreting displayed information, and using additional features.
   * Include any customization options and guidelines for script modification.
4. **Troubleshooting Section:**
   * Add a troubleshooting section addressing common issues and their resolutions.

### **Activity 6: Finalization**

1. **Final Testing:**
   * Perform comprehensive testing, including edge cases and error scenarios.
   * Confirm that the script operates as expected in various situations.
2. **Project Completion:**
   * Review the project against the initial scope and objectives.
   * Ensure that all features and functionalities are implemented successfully.
3. **Celebrate and Share:**
   * Share the finalized script with the community or colleagues.
   * Celebrate the completion of the Raspberry Pi System Monitor project!