Memory Check Alert Script

Create a shell script to check the available free memory on the system and alert the user if it falls below the threshold

Here's a simple shell script that checks the available free memory on a Linux system and alerts the user if it falls below a specified threshold.

Check Memory Alert

```
#!/bin/bash

# Set the threshold (in MB)
THRESHOLD=500  # Adjust as needed

# Get the available memory in MB
AVAILABLE_MEM=$(free -m | awk '/^Mem:/ {print $7}')

# Check if available memory is below the threshold
if [[ $AVAILABLE_MEM -lt $THRESHOLD ]]; then
    echo "Warning: Available memory is below ${THRESHOLD}MB! Current free memory: ${AVAI
    # You can add additional actions here, like sending an email or logging the event
fi
```

This script checks the available free memory and prints a warning if it falls below the specified threshold. You can modify it to send an email or log an event if needed. Let me know if you want enhancements! 🚀

Create a shell script to automate the creation of a new user with specific permissions and home directory.

Create User Script

```
#!/bin/bash

# Check if the script is run as root
if [[ $EUID -ne 0 ]]; then
    echo "This script must be run as root" >&2
    exit 1
fi

# Get username and home directory from user input
```

```
read -p "Enter the username: " USERNAME
read -p "Enter the home directory (default: /home/$USERNAME): " HOMEDIR
HOMEDIR=${HOMEDIR:-/home/$USERNAME}

# Create the user with the specified home directory
useradd -m -d "$HOMEDIR" -s /bin/bash "$USERNAME"

# Set password for the new user
passwd "$USERNAME"

# Assign user to a specific group
read -p "Enter the group name to assign (default: users): " GROUP
GROUP=${GROUP:-users}
usermod -aG "$GROUP" "$USERNAME"

# Set specific permissions for the home directory
chmod 750 "$HOMEDIR"
chown "$USERNAME:$GROUP" "$HOMEDIR"

echo "User $USERNAME has been created with home directory $HOMEDIR and added to group $G
```

This script automates the process of creating a new user, setting up their home directory, assigning them to a group, and setting the correct permissions. Let me know if you need any modifications! 🚀

↪ Asked ChatGPT

explain

The selected text `HOMEDIR=${HOMEDIR:-/home/$USERNAME}` is a Bash parameter expansion that sets the `HOMEDIR` variable. Here's how it works:

- `HOMEDIR` is assigned the value of itself (`$HOMEDIR`) **if it is already set**.
- If `HOMEDIR` is **not set or is empty**, it gets assigned the default value `/home/$USERNAME` .

This ensures that if the user does not provide a custom home directory, the system automatically assigns the default home directory `/home/$USERNAME` .

Write a shell script to find all large files greater than 1GB in a directory and move them to another directory.

Move Large Files

```
#!/bin/bash
```

```
# Check if the correct number of arguments is provided
if [[ $# -ne 2 ]]; then
    echo "Usage: $0 <source_directory> <destination_directory>"
    exit 1
fi

SOURCE_DIR=$1
DEST_DIR=$2

# Check if source directory exists
if [[ ! -d $SOURCE_DIR ]]; then
    echo "Error: Source directory $SOURCE_DIR does not exist."
    exit 1
fi

# Check if destination directory exists, create if not
if [[ ! -d $DEST_DIR ]]; then
    echo "Destination directory $DEST_DIR does not exist. Creating it..."
    mkdir -p "$DEST_DIR"
fi

# Find and move files larger than 1GB
find "$SOURCE_DIR" -type f -size +1G -exec mv {} "$DEST_DIR" \;

echo "All files larger than 1GB have been moved from $SOURCE_DIR to $DEST_DIR."
```

This script finds and moves all files larger than 1GB from a specified source directory to a destination directory. Let me know if you need any modifications! 🚀

↪ Asked ChatGPT

explain

The selected text `-exec mv {} "$DEST_DIR" \;` is part of the `find` command and is used to move files that match the search criteria. Here's a breakdown of what it does:

- `find "$SOURCE_DIR" -type f -size +1G`: Searches for files (`-type f`) larger than 1GB (`-size +1G`) in the source directory.
- `-exec mv {} "$DEST_DIR" \;`:
    - `-exec` allows executing a command (`mv` in this case) on each file found.
    - `{}` is a placeholder that represents the file found by `find`.
    - `"$DEST_DIR"` is the target directory where the files will be moved.

- `\;` marks the end of the `-exec` command.

This ensures that every file larger than 1GB is moved from `SOURCE_DIR` to `DEST_DIR`. Let me know if you need further clarification!

Write a shell script that automatically updates all installed packages on a system and reboots the system if needed.

Update Packages Reboot

```
#!/bin/bash

# Check if the script is run as root
if [[ $EUID -ne 0 ]]; then
    echo "This script must be run as root" >&2
    exit 1
fi

# Update package list and upgrade all packages
echo "Updating package list and upgrading packages..."
apt update && apt upgrade -y

# Check if a reboot is required
if [[ -f /var/run/reboot-required ]]; then
    echo "A reboot is required. Rebooting now..."
    reboot
else
    echo "System update complete. No reboot required."
fi
```

This script updates all installed packages on a Debian-based system and checks if a reboot is required. If needed, it automatically reboots the system. Let me know if you need modifications! 🚀

↪ Asked ChatGPT

explain

The file `/var/run/reboot-required` is a system-generated file found on Debian-based Linux distributions (like Ubuntu). It is created by the package manager when a system update includes changes that require a reboot (such as kernel updates or critical system components).

In this script:

- `[[ -f /var/run/reboot-required ]]` checks if the file exists.
- If the file exists, the script prints a message and reboots the system.
- If the file does not exist, the script confirms that no reboot is necessary.

This ensures the system stays updated and applies necessary changes without manual intervention.

Write a script to count the number of lines in all .log files in a specified directory.

Count Log Lines

```
#!/bin/bash

# Check if the correct number of arguments is provided
if [[ $# -ne 1 ]]; then
    echo "Usage: $0 <directory>"
    exit 1
fi

DIR=$1

# Check if the directory exists
if [[ ! -d $DIR ]]; then
    echo "Error: Directory $DIR does not exist."
    exit 1
fi

# Count the number of lines in each .log file and display the total
TOTAL_LINES=0
for FILE in "$DIR"/*.log; do
    if [[ -f $FILE ]]; then
        LINES=$(wc -l < "$FILE")
        echo "$FILE: $LINES lines"
        TOTAL_LINES=$((TOTAL_LINES + LINES))
    fi
done

echo "Total lines in all .log files: $TOTAL_LINES"
```

This script counts the number of lines in each `.log` file within a specified directory and provides a total count. Let me know if you need modifications! 🚀

↪ Asked ChatGPT

explain

The selected text `for FILE in "$DIR"/*.log; do` is a Bash loop that iterates over all `.log` files in the specified directory (`$DIR`). Here's what it does:

- `"$DIR"/*.log`: Expands to match all files with a `.log` extension in the given directory.
- `for FILE in ...; do`: Starts a loop, where each matched file is assigned to the variable `FILE` in turn.
- The subsequent commands inside the loop operate on each `FILE`.

This ensures that the script processes every `.log` file found in the directory, counting its lines and accumulating the total count.

↪ Asked ChatGPT

explain

The selected text `wc -l < "$FILE"` counts the number of lines in the file specified by `"$FILE"`. Here's how it works:

- `wc -l`: The `wc` (word count) command with the `-l` option counts the number of lines in the input.
- `< "$FILE"`: This redirects the contents of `"$FILE"` to `wc -l`, ensuring the file is read correctly.

This approach prevents unnecessary file listing in the output, making it a cleaner way to count lines compared to `wc -l "$FILE"`, which would also print the filename.

Write a script that checks for the presence of specific software on the system on the system (e.g. Docker, Git) and installs if it is missing.

Check Install Software

```
#!/bin/bash

# List of required software
```

```
SOFTWARE_LIST=(docker git)

# Function to check and install software
install_software() {
    local software=$1
    if ! command -v "$software" &> /dev/null; then
        echo "$software is not installed. Installing..."
        apt update && apt install -y "$software"
    else
        echo "$software is already installed."
    fi
}

# Ensure the script is run as root
if [[ $EUID -ne 0 ]]; then
    echo "This script must be run as root" >&2
    exit 1
fi

# Iterate through the software list and install if missing
for software in "${SOFTWARE_LIST[@]}"; do
    install_software "$software"
done

echo "Software check and installation completed."
```

This script checks for the presence of Docker and Git on the system, installing them if they are missing. Let me know if you need additional functionality! 🚀

↪ Asked ChatGPT

explain

The selected text `if ! command -v "$software" &> /dev/null;` checks whether a specific software command is available on the system. Here's how it works:

- `command -v "$software"`: This checks if the specified software (command) exists in the system's executable paths.
- `&> /dev/null`: Redirects both standard output (`stdout`) and standard error (`stderr`) to `/dev/null`, preventing any output from appearing.
- `if ! ...; then`: The `!` negates the condition, so the block inside `then` executes **only if** the command is **not found**, meaning the software is missing.

This ensures the script only installs software that is not already present. Let me know if you need further clarification! 🚀

can we use "which" instead?

Yes, you can use `which` instead of `command -v`, but `command -v` is generally the preferred method for checking if a command exists in modern scripts. Here's why:

Differences Between `which` and `command -v`:

1. **Portability**
   - `command -v` is a shell built-in and works across different shells (Bash, Zsh, Dash, etc.).
   - `which` is an external command and may not be available in minimal environments.
2. **Accuracy**
   - `command -v` returns the exact path or a shell function if the command exists.
   - `which` only searches for executables in `$PATH`, so it may not detect shell functions or aliases.
3. **Performance**
   - `command -v` is faster since it doesn't spawn a separate process like `which`.

How to Modify Your Script to Use `which`

If you still prefer `which`, update the check like this:

```bash
if ! which "$software" &> /dev/null; then
```

This will still work, but `command -v` is the more robust choice. Let me know if you want to proceed with `which`! 🚀