Tutorial: Automating System Maintenance with `system_maintenance.sh`

Maintaining a healthy Linux system can be time-consuming if done manually, especially when it involves clearing caches, updating packages, and removing unnecessary files. To simplify this process, I've created a script that automates crucial maintenance tasks such as fetching the best mirrors, upgrading the system, cleaning the package cache, and removing orphaned or broken packages.

This tutorial will guide you through setting up the `system_maintenance.sh` script, integrating it into your workflow, and scheduling it for automatic execution.

Overview of the Script

The `system_maintenance.sh` script automates the following tasks:

1. Fetches the best mirrors for faster downloads.
2. Upgrades the system using the package manager.
3. Cleans the package cache.
4. Removes orphaned packages and broken packages.
5. Sets up a vacuum for journal logs to ensure they don't fill up your storage.
6. Removes cached files from uninstalled packages.

This setup can be easily integrated into your workflow, and I'll show you how to automate it so your system stays optimized without any manual effort.

Prerequisites

Before running the script, you need to ensure the following:

- **Tools**: You'll need `wmctrl` and `alacritty` for managing the terminal and workspace setup.

  You can install them with the following command:

  ```
  sudo pacman -S wmctrl alacritty
  ```

- **Basic Knowledge**: You should be familiar with the `.bashrc` file and the Pacman package manager, as this tutorial will touch on editing the `.bashrc` file and running Pacman commands.

The Maintenance Script

Here's the `system_maintenance.sh` script that performs all the tasks we need:

```bash
#!/bin/bash

# Function to run commands with high priority
run_with_priority() {
    sudo ionice -c 2 -n 0 nice -n -20 "$@"
}

# Step 1: Clean Pacman cache
echo "Cleaning pacman cache..."
run_with_priority sudo pacman -Scc --noconfirm

# Step 2: Remove orphaned packages
echo "Removing orphaned packages..."
orphans=$(pacman -Qtdq 2>/dev/null)
if [ ! -z "$orphans" ]; then
    run_with_priority sudo pacman -Rns $orphans --noconfirm
else
    echo "No orphaned packages found."
fi

# Step 3: Check and remove broken packages
echo "Checking for broken packages..."
broken_packages=$(pacman -Qk 2>/dev/null | grep " is missing" | awk
        '{print $1}')
if [ ! -z "$broken_packages" ]; then
    echo "Removing broken packages..."
    run_with_priority sudo pacman -Rns $broken_packages --noconfirm
else
    echo "No broken packages found."
fi

# Step 4: Refresh Pacman keys
echo "Refreshing pacman keys..."
run_with_priority sudo pacman-key --init
run_with_priority sudo pacman-key --populate archlinux

# Step 5: Fetch the best mirrors
echo "Fetching the best mirrors..."
run_with_priority sudo reflector --country United-States --country
        Canada --latest 20 -p https --sort rate --save
        /etc/pacman.d/mirrorlist

# Step 6: Update system and installed packages using pacman
echo "Updating system and installed packages..."
run_with_priority sudo pacman -Syyu --noconfirm

echo "All tasks completed successfully!"
```

Key Functions in the Script:

1. **`run_with_priority()`**: This function ensures all the commands run with high priority by using `ionice` and `nice`, which ensures that the system processes the maintenance tasks quickly and efficiently.
2. **Pacman Cache Clean-Up**: This removes unused cached packages that can take up unnecessary space.
3. **Removing Orphaned and Broken Packages**: Orphaned packages (packages that are no longer required by any installed package) are removed to keep the system clean. Similarly, broken packages (packages missing required files) are cleaned up.
4. **Refreshing Pacman Keys**: This ensures that your system's package manager has valid and updated keys to verify packages.
5. **Fetching Mirrors**: The `reflector` command fetches the best mirrors based on your location, ensuring faster package downloads.
6. **System Update**: Finally, it updates your system and all installed packages.

Step-by-Step Guide to Automating the Script

Step 1: Add the Script to Your `.bashrc`

To run the maintenance tasks in the background or on a specific workspace, we'll modify the `.bashrc` file and add our script.

1. Open your `.bashrc` file with a text editor:

   ```
   nano ~/.bashrc
   ```

2. Add the following lines to the bottom of the file:

   ```
   # Switch to workspace 6 and open Alacritty to run system
           maintenance
   wmctrl -s 5
   alacritty -e /path/to/your/actual/system_maintenance.sh
   ```

   This sets up the workspace switch using `wmctrl` and opens `alacritty` to run the script in a new terminal on workspace 6.

3. Save and close the file.

Step 2: Make the Script Executable

You need to ensure the script has executable permissions:

```
chmod +x /path/to/system_maintenance.sh
```

This makes the script runnable from the terminal or any automated process like cron.

Step 3: Set Up a Cron Job for Weekly Maintenance

We'll now schedule the script to run every Sunday at midnight:

1. Open your crontab file:

   ```
   crontab -e
   ```

2. Add the following line to schedule the script to run weekly:

   ```
   0 0 * * 0 /path/to/system_maintenance.sh
   ```

This will execute the script at midnight every Sunday, automating system maintenance without needing manual intervention.

Conclusion

By following this setup, you ensure that your system is regularly maintained with minimal effort. The `system_maintenance.sh` script handles essential tasks such as cleaning the package cache, removing orphaned packages, and keeping your mirrors and packages up to date.

Setting up the cron job and integrating the script with `.bashrc` guarantees that maintenance runs seamlessly in the background, helping keep your system optimized without manual input.