

Maintenance

- [Release/Clear cache memory RAM](#)
- [Large files linux](#)
- [Find hidden process running](#)
- [IP Tables](#)
- [Buzz noise audio - KDE](#)
- [epomaker keyboard detected as apple](#)

Release/Clear cache memory RAM

Every Linux System has three options to clear cache without interrupting any processes or services.

1. Clear PageCache only.

```
# sync; echo 1 > /proc/sys/vm/drop_caches
```

2. Clear dentries and inodes.

```
# sync; echo 2 > /proc/sys/vm/drop_caches
```

3. Clear pagecache, dentries, and inodes.

```
# sync; echo 3 > /proc/sys/vm/drop_caches
```

Explanation of the above command.

sync will flush the file system buffer. Command Separated by `“;”` run sequentially. The shell waits for each command to terminate before executing the next command in the sequence. As mentioned in the kernel documentation, writing to **drop_cache** will clean cache without killing any application/service, command echo is doing the job of writing to file.

If you have to clear the disk cache, the first command is safest in enterprise and production as `“...echo 1 > ...”` will clear the **PageCache** only. It is not recommended to use the third option above `“...echo 3 >”` in production until you know what you are doing, as it will clear **pagecache, dentries, and inodes**.

Is it a good idea to free Buffer and Cache in Linux that might be used by Linux Kernel?

Free Buffer and Cache in Linux

When you are applying various settings and want to check, if it is actually implemented specially on the I/O-extensive benchmark, then you may need to clear the buffer cache. You can drop cache as explained above without rebooting the System i.e., no downtime required.

Linux is designed in such a way that it looks into the disk cache before looking onto the disk. If it finds the resource in the cache, then the request doesn't reach the disk. If we clean the cache, the disk cache will be less useful as the OS will look for the resource on the disk.

Moreover, it will also slow the system for a few seconds while the cache is cleaned and every resource required by OS is loaded again in the disk cache.

Now we will be creating a shell script to auto clear RAM cache daily at **2 am** via a cron scheduler task. Create a shell script **clearcache.sh** and add the following lines.

```
#!/bin/bash
# Note, we are using "echo 3", but it is not recommended in production instead use "echo 1"
echo "echo 3 > /proc/sys/vm/drop_caches"
```

Set execute permission on the **clearcache.sh** file.

```
# chmod 755 clearcache.sh
```

Now you may call the script whenever you are required to clear the ram cache.

Now set a cron to clear RAM cache every day at 2 am. Open **crontab** for editing.

```
# crontab -e
```

Append the below line, save and exit to run it at **2 am** daily.

```
0 2 * * * /path/to/clearcache.sh
```

For more details on how to cron a job, you may like to check our article on [11 Cron Scheduling Jobs](#).

Is it a good idea to auto clear the RAM cache on the production server?

Clear RAM Cache on Linux Production Server?

No! it is not. Think of a situation when you have scheduled the script to clear ram cache every day at **2 am**. Every day at 2 am the script is executed and it flushes your RAM cache. One day for whatsoever reason may be more than expected users are online on your website and seeking resources from your server.

At the same time, the scheduled script runs and clears everything in the cache. Now all the users are fetching data from the disk. It will result in a server crash and corrupt the database. So clear ram-cache only when required, and known your footsteps, else you are a Cargo Cult System Administrator.

How to Clear Swap Space in Linux?

If you want to clear Swap space, you may like to run the below command.

```
# swapoff -a && swapon -a
```

Also, you may add the above command to a cron script above, after understanding all the associated risks.

Now we will be combining both above commands into one single command to make a proper script to clear RAM Cache and Swap Space.

```
# echo 3 > /proc/sys/vm/drop_caches && swapoff -a && swapon -a && printf '\n%s\n' 'Ram-cache and Swap Cleared'
```

OR

```
$ su -c "echo 3 > /proc/sys/vm/drop_caches" && swapoff -a && swapon -a && printf '\n%s\n' 'Ram-cache and Swap Cleared'
```

After testing both the above commands, we will run the command “**free -h**” before and after running the script and will check the cache.

Clear RAM Cache and Swap Space

Large files linux

df -h shows hard drive space

```
sudo du -a /dir/ | sort -n -r | head -n 20
```

this will list larger files/folders in /dir

```
find / -type f -size +20000k -exec ls -lh {} \; 2> /dev/null | awk '{ print $NF ": " $5 }' | sort -nrk 2,2
```

this will find files larger than 20000k

Find hidden process running

```
ls -l | grep txt | grep "\."
```

IP Tables

to list all ip tables by chain

```
iptables -L --line-numbers
```

add iptable on line

```
sudo iptables -I INPUT 1 -p tcp -s 10.190.195.123 --dport 22 -j ACCEPT
```

```
sudo iptables -I INPUT 2 -p tcp -s 10.190.227.123 --dport 22 -j ACCEPT
```

```
sudo iptables -I INPUT 3 -p tcp -s 10.190.195.123 --dport 8443 -j ACCEPT
```

```
sudo iptables -I INPUT 4 -p tcp -s 10.190.227.123 --dport 8443 -j ACCEPT
```

```
sudo iptables -I INPUT 5 -p tcp --dport 22 -j REJECT
```

```
sudo iptables -I INPUT 6 -p tcp --dport 8443 -j REJECT
```

delete existing chain line

```
iptables -D INPUT 2
```

this will delete rule 2

to save the iptables

```
service iptables save
```

Buzz noise audio - KDE

Change your working directory to /etc/modprobe.d:

```
cd /etc/modprobe.d
```

And now, create a new file named `audio_disable_powersave.conf` and open with the nano text editor using the given command:

```
sudo nano audio_disable_powersave.conf
```

And put the following lines in that file to turn off the power-saving setting in the sound card permanently:

```
options snd_hda_intel power_save=0
```

fix buzzing sound in linux

For a **USB sound card**, you can use `snd_usb_audio`:

```
options snd_usb_audio power_save=0
```

Now, save changes and exit the Nano text editor by pressing Ctrl+X keys. Reboot your system, and you can enjoy a noise-free workspace.

epomaker keyboard detected as apple

Epomaker Galaxy80 keyboard

The Problem

Recently, my Keychron K8 keyboard has died (PCB [\[1\]](#)), so I started hunting for a new keyboard. Another Keychron was out of the question, so I decided to give Epomaker a try. I stumbled into a B-stock [Epomaker Galaxy80](#) and decided to give it a shot. I am very happy with the feel of the keyboard, typing feels much better than the K8, at least to me.

However, I ran into a problem in linux (Fedora Workstation, although this is not distro-specific): I was unable to use the F-row (F1 - F12), so that Alt + F4 would not close the window, F11 would not trigger full screen etc. When I used the online [keyboard testers](#), it would register keys/commands such as “Files”, “Calculator”, “Volume Up/Down” etc.:

“F-row keypresses in a keyboard tester”

I searched the keyboard manual, thinking it must be some sort of an Fn lock being activated, but there does not seem to be a feature like that on my keyboard. Even worse, using the Fn key + one of the F-row keys was not triggering the actual F-row key either. I had no (useful) way to use my F-row keys!

This issue has been reported with many other Epomaker keyboards, such as EP84, TH80, RT80, RT100 or GMK87, but also with other brands (including some Keychrons). The good news is, the fix should be the same for all of them!

The Solution

After some furious googling, I actually found the solution to the problem, which turns out to be on the system side rather than in the keyboard itself. Simply [reconfiguring](#) your [HID Apple module](#) fixes the issue. If you're technically savvy and/or do not care about any further details, go use the module configuration and enjoy using your keyboard.

```
echo "options hid_apple fnmode=2" | sudo tee /etc/modprobe.d/hid_apple.conf  
sudo dracut --regenerate-all --force  
# reboot and enjoy your fully functioning keyboard!
```

(courtesy of Reddit user [Outrageous_Opinion84](#) and Joao-Peterson on GitHub)

The Solution Explained

Below, I would like to go into more details for several reasons:

1. Reddit is going to paywall some of the content, and other content might get authwalled - therefore the solution may not be readily available.
2. It took me some time to find the solution, so by writing this article, I'm hoping the more elaborate explanation will be more SEO-friendly than some of the existing answers scatted over the internet.
3. I am sure I will forget this solution next time I install a new system and writing it up will hopefully mean I will not need to re-figure it out.
4. I wanted to learn (and share) in a bit more detail what happens with the configuration files and how the fix works.

First, let's take a look at how the Epomaker keyboard reports to the system when loaded up (the last but one row):

output of lsusb

Now the problem is that the keyboard is being detected as an Apple wireless keyboard clone - an abomination of a keyboard discontinued in 2015:

output of sudo dmesg

By default, the F-row keys on the Apple wireless keyboard emit the media commands, which is how those keypresses will get interpreted by the `hid_apple` kernel module. Luckily, the module offers configurable function modes (`fnmode`), as documented on [Arch Wiki](#) or [Ubuntu Wiki](#). We are interested in the `fnmode=2` option:

- 2 - normally function keys, switchable to media keys by holding Fn key (=auto on non-Apple keyboards)

You can temporarily test the parameter by sending the `fnmode` parameter to the kernel module via sysfs:

(You likely need to do a `sudo su` first)

```
echo 2 >> /sys/module/hid_apple/parameters/fnmode
```

Now your keyboard should work as **GodEpomaker** intended! The problem is that this change will not be persisted after reboot.

The way to do it is through the `modprobe.d` configuration files, which allow to pass parameters to the modules when they are loaded during system boot.

The following command will create the configuration file for the `hid_apple` module and pass the `fnmode=2` parameter to it:

```
echo "options hid_apple fnmode=2" | sudo tee /etc/modprobe.d/hid_apple.conf
```

Now to apply the changes to the initramfs (a mini-image with your kernel, including its modules), you need to run the `dracut` command to re-generate initramfs.

```
sudo dracut --regenerate-all --force
```

Note: Some distributions may require a different command to re-generate initramfs, such as `sudo update-initramfs -u` for Ubuntu or `sudo mkinitcpio -P` in Arch.

Finally, all you need to do is either reboot your machine, or reload the module:

```
sudo rmmod hdi_apple  
modprobe hdi_apple
```