Name : Shravya Mohan

Class : 3A

USN : ENG24CY0049

**LINUX PROGRAMMING ASSIGNMENT 10**

*DETAILED ANALYSIS OF “LSUSB” COMMAND*

The lsusb command in Linux is a utility used to display information about USB buses and the devices connected to them. The name itself is straightforward because it stands for “List USB”. It is included as part of the usbutils package, a collection of tools created to help users and administrators inspect and troubleshoot USB hardware. As USB devices became more common in Linux systems during the early 2000s, there arose a need for a standard and reliable method to check which USB devices were detected by the operating system. This led to the development of utilities such as lsusb.

The original author of the lsusb command is **Thomas Sailer**. His name appears in the manual page and he wrote the early versions of the command. Thomas Sailer contributed to many USB-related tools during a period when Linux was still building its hardware support. Even after the initial development, other open-source contributors continued to improve and maintain the command.

Another important figure related to the broader USB subsystem in Linux is **Greg Kroah-Hartman**, a well-known Linux kernel developer. Although he did not write lsusb itself, he has played a major role in maintaining the Linux USB subsystem, device drivers, and many hardware-related parts of the kernel. Because of this, both Thomas Sailer (as the command’s author) and Greg Kroah-Hartman (as a long-term maintainer of the USB subsystem) are associated with the history and evolution of USB handling in Linux. The usbutils package, which contains lsusb, is the result of continuous teamwork across the Linux community.

Over time, lsusb has become a standard tool for identifying USB hardware, diagnosing issues, and confirming driver status.

For system administrators, USB devices are not limited to storage devices like pen drives. Many essential computer peripherals operate through USB, such as keyboards, mice, Bluetooth modules, network adapters, cameras, audio interfaces, Wi-Fi dongles, and various sensors. When any of these devices malfunction, the first step is often to confirm whether the system recognises them. This is where lsusb becomes invaluable. By displaying vendor IDs, product IDs, bus numbers, and device numbers, the command provides immediate clarity about which devices are connected and how the system is interpreting them.

lsusb also assists administrators working in virtualised environments. When using tools such as VMware, VirtualBox, or QEMU, USB passthrough is a common requirement, and lsusb helps verify whether the virtual machine is receiving access to the hardware from the host system. It also helps with hardware audits, documentation, and identifying potential issues such as insufficient power or limited bandwidth on a particular USB hub. In essence, the command allows administrators to “see what the kernel sees”, making it an essential troubleshooting aid.

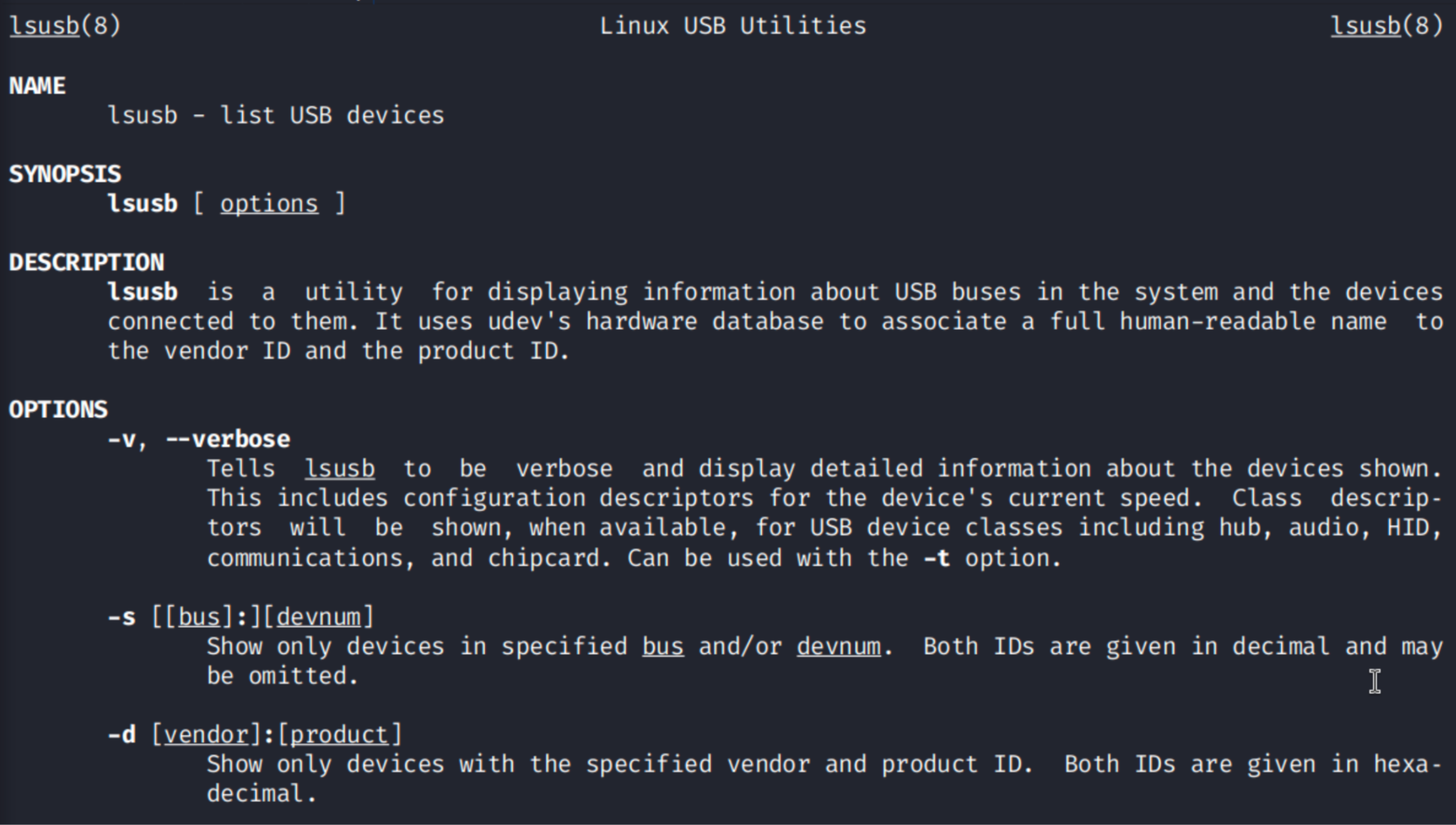
The way lsusb works internally is quite straightforward. It does not communicate directly with the USB hardware. Instead, it gathers information stored by the Linux kernel. The kernel continuously updates specific directories whenever a USB device is connected, disconnected, or reconfigured. These directories include /sys/bus/usb/devices/ and /dev/bus/usb/. They contain structured data about each USB device, including its vendor ID, product ID, device class, driver details, and the port or bus to which it is connected. The lsusb command simply reads and formats this information so that it is easily understandable for the user.

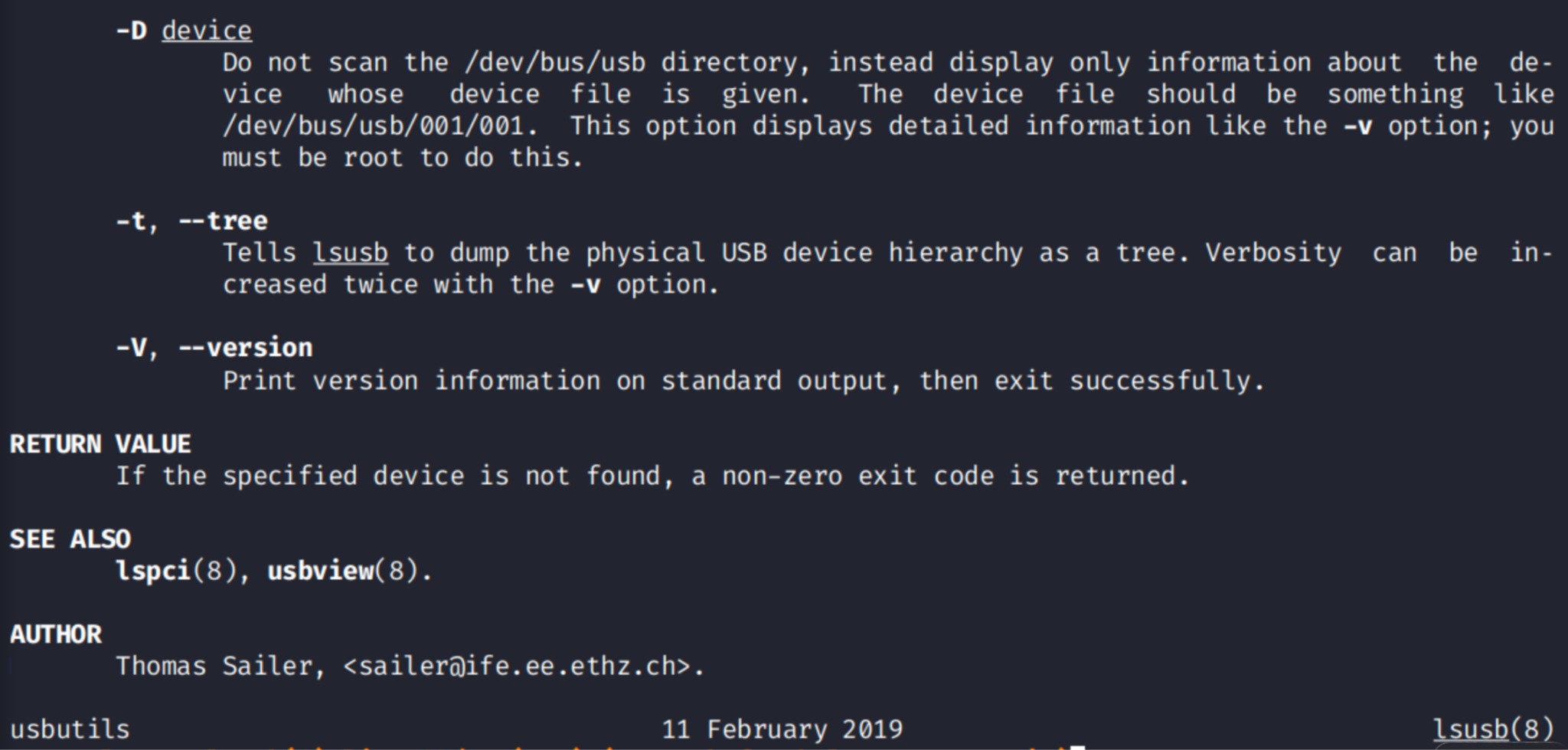
A typical output from lsusb includes entries representing the system’s built-in USB controllers and the devices attached to them. For example, the lines showing “Linux Foundation 2.0 root hub” or “Linux Foundation 3.0 root hub” represent the system’s USB controllers. These are the root hubs, meaning they are the primary controllers responsible for managing USB ports. USB 2.0 and USB 3.0 indicate the speeds supported by each controller.



In some environments, especially virtualised ones or systems running macOS as the host, lsusb may show virtual USB devices instead of physical ones. These include entries such as “Apple Virtual USB Keyboard”, “Apple Virtual USB Digitizer”, or “Apple Virtual USB Mass Storage Device”. These devices are not physically connected hardware. Instead, they are virtual interfaces created by the host operating system or the virtual machine manager. They allow the guest operating system to interact with host input devices or storage as if they were actual USB hardware.

For anyone needing full technical documentation or a complete list of flags and options, the manual page for the command can be accessed with man lsusb. This provides a deeper look into advanced features such as verbose output, tree views of the USB hierarchy, device-specific queries, and driver-level information.





To sum it up, lsusb is a simple yet powerful diagnostic tool within Linux. Its ability to clearly present how the kernel interprets USB devices makes it essential for both everyday use and more advanced system administration tasks. Its development reflects the collaborative nature of open-source software, and its continued relevance highlights the central role of USB technology in modern computing.

*DETAILED ANALYSIS OF “TASKSET” COMMAND*

The taskset command is a Linux utility used to set or retrieve the CPU affinity of a process. CPU affinity refers to the specific CPU cores on which a process is allowed to run. In simple terms, it lets you decide which processor core a particular programme should use, rather than letting the operating system automatically choose it. This can be helpful in improving performance, managing system load and controlling how resources are shared between processes.

The history of the taskset command is linked to the growth of multi-core processors. As computers began moving from a single CPU to multiple CPU cores, Linux needed tools that could control how tasks were distributed across these cores. The command belongs to the util-linux package, which collects many core Linux utilities. The original author credited for taskset is **Robert Love**, a well known Linux developer and author. His work in process scheduling and kernel performance helped shape tools that interact closely with CPU handling. Robert Love wrote parts of util-linux related to process management, which is why his name appears associated with taskset.

As multi-core CPUs became standard in the early 2000s, Linux users needed a straightforward command that could let them pin processes to certain cores or check how the system was scheduling tasks. This is where taskset became useful. It allowed developers and system administrators to test performance, reduce CPU interference between tasks, and keep certain workloads separate. Over time, the command has become a reliable tool for performance tuning and system control.

System administrators use taskset for several reasons. When running heavy applications, it may be useful to tell the system to run them on specific cores so that they do not interfere with other critical processes. For example, a database server might be assigned to one set of cores while background tasks run on another. This avoids them competing for the same CPU resources. In virtualisation environments, controlling CPU usage becomes even more important, and taskset helps administrators ensure that each virtual machine or container receives the correct CPU allocation.

The internal working of taskset is quite simple. The command interacts with the Linux kernel’s scheduler to apply a bitmask that represents which CPU cores are permitted for a process. Every core has a numerical position in the bitmask and taskset either reads the current mask or applies a new one. The kernel then ensures the process only runs on those selected cores.

taskset can also be used when launching a new programme, which means the CPU affinity can be set before the process even begins running.

A usage of the command:

taskset -c 0,2 myprogram

This would force the programme to run only on CPU cores 0 and 2.

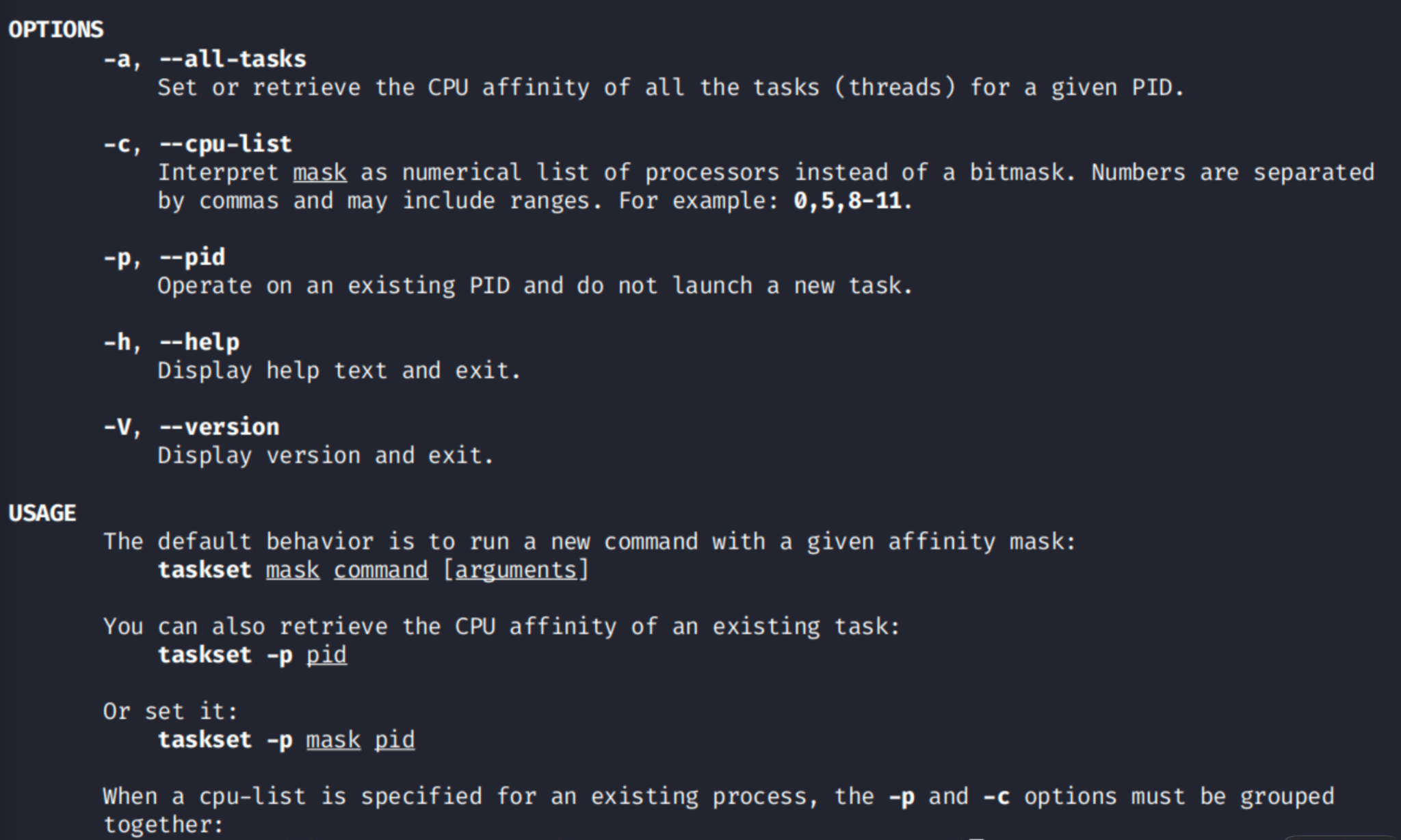
Another example:

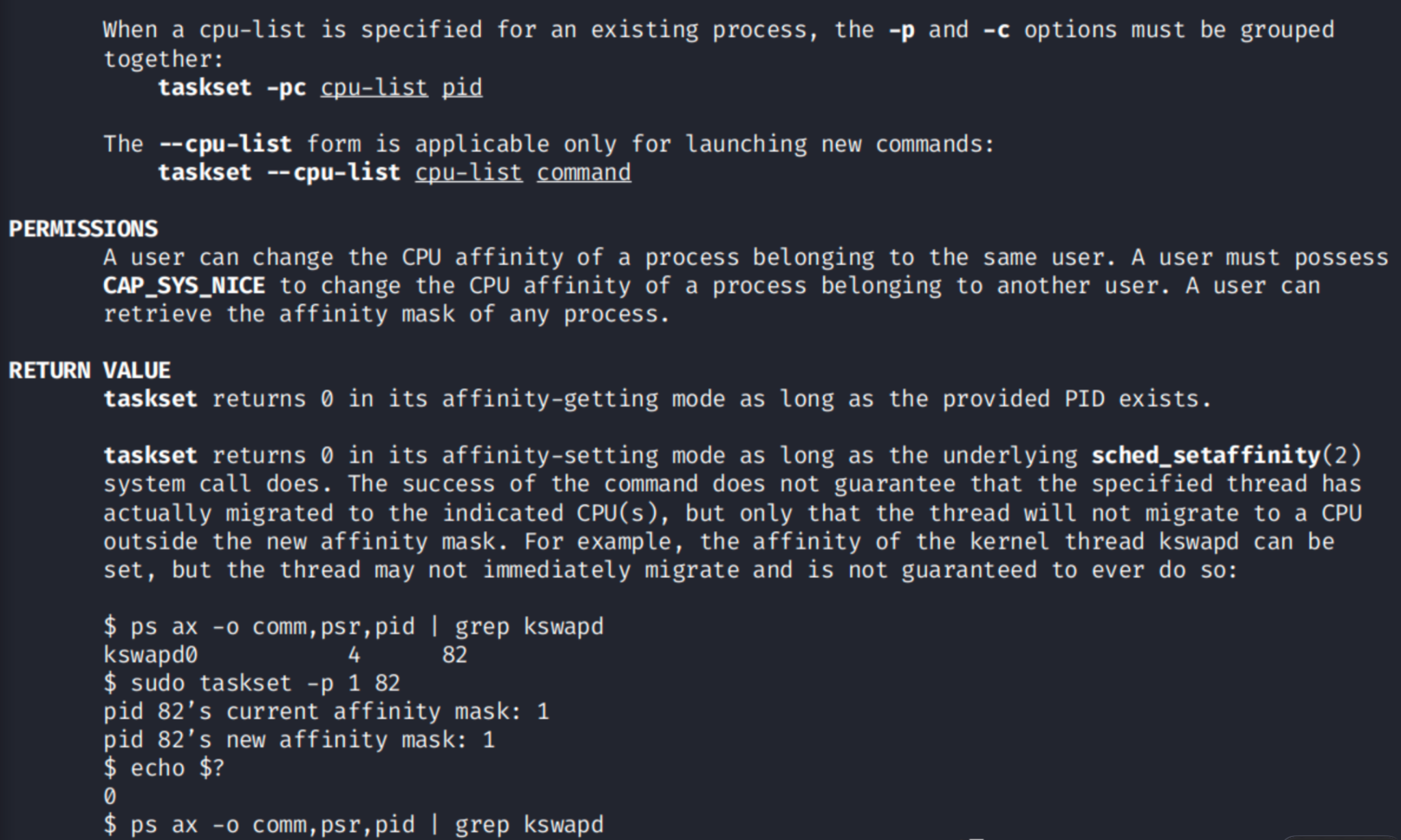
taskset -p 1234

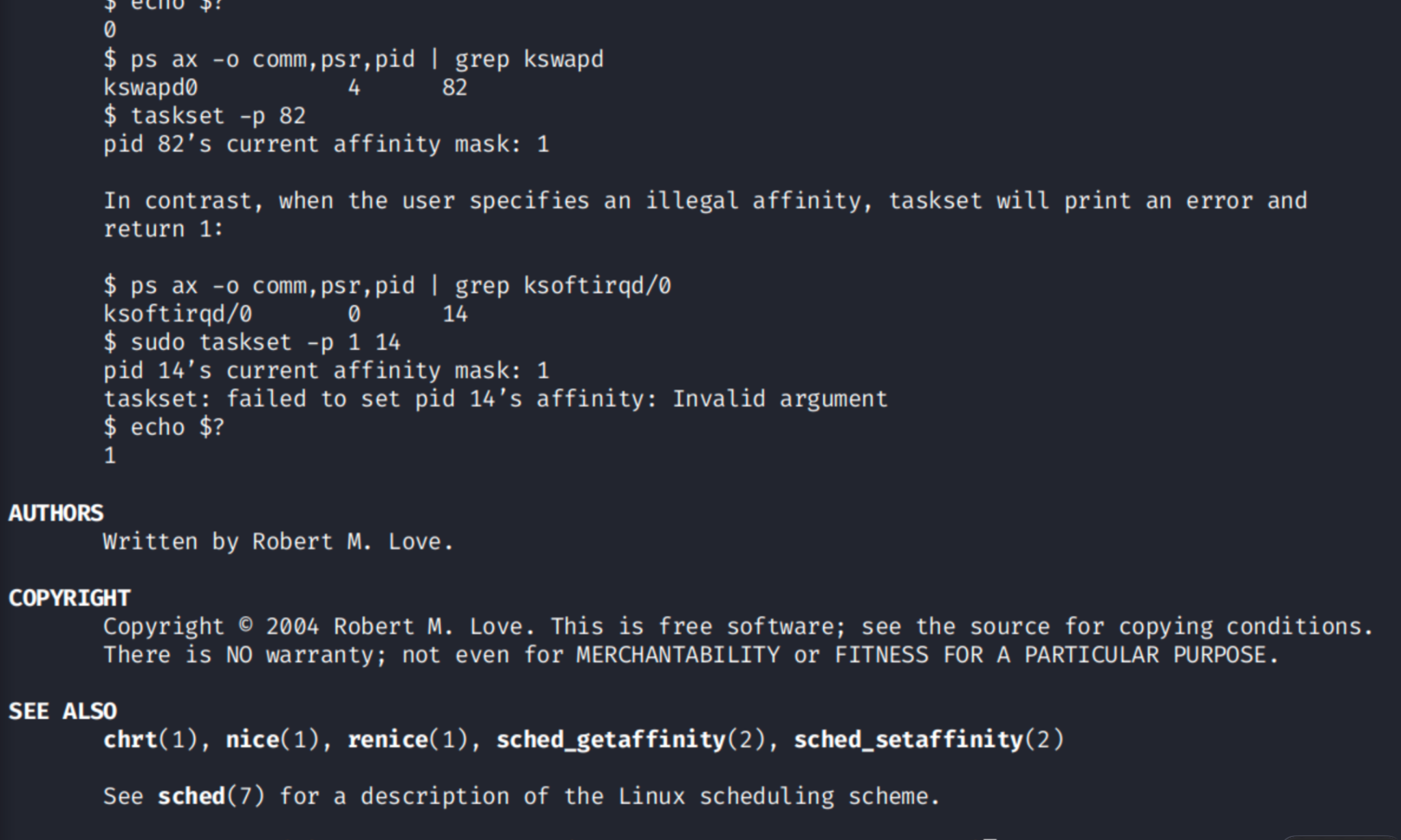
This displays the existing CPU affinity of a process with process ID 1234.

Because the command works closely with the kernel’s scheduler, it provides a reliable way to influence how the system uses CPU resources. This makes it a valuable tool for load balancing, testing performance, and managing resource-heavy tasks.

Like other Linux utilities, the complete documentation for taskset can be found using the man command (for flags):



**

**