<https://learning.lpi.org/en/learning-materials/101-500/101/101.1/101.1_01/>

1. [Topic 101: System Architecture](https://learning.lpi.org/en/learning-materials/101-500/101/)
2. [101.1 Determine and configure hardware settings](https://learning.lpi.org/en/learning-materials/101-500/101/101.1/)
3. [101.1 Lesson 1](https://learning.lpi.org/en/learning-materials/101-500/101/101.1/101.1_01/)

# 101.1 Lesson 1

|  |  |
| --- | --- |
| **Certificate:** | LPIC-1 |
| **Version:** | 5.0 |
| **Topic:** | 101 System Architecture |
| **Objective:** | 101.1 Determine and configure hardware settings |
| **Lesson:** | 1 of 1 |

## Introduction

Since the early years of electronic computing, business and personal computer manufacturers have integrated a variety of hardware parts in their machines, which in turn need to be supported by the operating system. That could be overwhelming from the operating system developer’s perspective, unless standards for instruction sets and device communication are established by the industry. Similar to the standardized abstraction layer provided by the operating system to an application, these standards make it easier to write and maintain an operating system that is not tied to a specific hardware model. However, the complexity of the integrated underlying hardware sometimes requires adjustments on how resources should be exposed to the operating system, so it can be installed and function correctly.

Some of these adjustments can be made even without an installed operating system. Most machines offer a configuration utility that can be executed as the machine is turned on. Until the mid 2000s, the configuration utility was implemented in the BIOS (Basic Input/Output System), the standard for firmware containing the basic configuration routines found in x86 motherboards. From the end of the first decade of the 2000s, machines based on the x86 architecture started to replace the BIOS with a new implementation called UEFI (Unified Extensible Firmware Interface), which has more sophisticated features for identification, testing, configuration and firmware upgrades. Despite the change, it is not uncommon to still call the configuration utility BIOS, as both implementations fulfill the same basic purpose.

|  |  |
| --- | --- |
| **Note** | Further details on the similarities and differences between BIOS and UEFI will be covered in a later lesson. |

### Device Activation

The system configuration utility is presented after pressing a specific key when the computer is turned on. Which key to press varies from manufacturer to manufacturer, but usually it is Del or one of the function keys, such as F2 or F12. The key combination to use is often displayed in the power on screen.

In the BIOS setup utility it is possible to enable and disable integrated peripherals, activate basic error protection and change hardware settings like IRQ (interrupt request) and DMA (direct memory access). Changing these settings is rarely needed on modern machines, but it may be necessary to make adjustments to address specific issues. There are RAM technologies, for example, that are compatible with faster data transfer rates than the default values, so it is recommended to change it to the values specified by the manufacturer. Some CPUs offer features that may not be required for the particular installation and can be deactivated. Disabled features will reduce power consumption and can increase system protection, as CPU features containing known bugs can also be disabled.

If the machine is equipped with many storage devices, it is important to define which one has the correct bootloader and should be the first entry in the device boot order. The operating system may not load if the incorrect device comes first in the BIOS boot verification list.

### Device Inspection in Linux

Once devices are correctly identified, it is up to the operating system to associate the corresponding software components required by them. When a hardware feature is not working as expected, it is important to identify where exactly the problem is happening. When a piece of hardware is not detected by the operating system, it is most likely that the part — or the port to which it is connected — is defective. When the hardware part is correctly detected, but still does not properly work, there may be a problem on the operating system side. Therefore, one of the first steps when dealing with hardware-related issues is to check if the operating system is properly detecting the device. There are two basic ways to identify hardware resources on a Linux system: to use specialized commands or to read specific files inside special filesystems.

#### Commands for Inspection

The two essential commands to identify connected devices in a Linux system are:

lspci

Shows all devices currently connected to the PCI (Peripheral Component Interconnect) bus. PCI devices can be either a component attached to the motherboard, like a disk controller, or an expansion card fitted into a PCI slot, like an external graphics card.

lsusb

Lists USB (Universal Serial Bus) devices currently connected to the machine. Although USB devices for almost any imaginable purpose exist, the USB interface is largely used to connect input devices — keyboards, pointing devices — and removable storage media.

The output of commands lspci and lsusb consists of a list of all PCI and USB devices identified by the operating system. However, the device may not be fully operational yet, because every hardware part requires a software component to control the corresponding device. This software component is called a kernel module and it can be part of the official Linux kernel or added separately from other sources.

Linux kernel modules related to hardware devices are also called drivers, as in other operating systems. Drivers for Linux, however, are not always supplied by the device’s manufacturer. Whilst some manufacturers provide their own binary drivers to be installed separately, many drivers are written by independent developers. Historically, parts that work on Windows, for example, may not have a counterpart kernel module for Linux. Nowadays, Linux-based operating systems have strong hardware support and most devices work effortlessly.

Commands directly related to hardware often require root privileges to execute or will only show limited information when executed by a normal user, so it may be necessary to login as root or to execute the command with sudo.

The following output of command lspci, for example, shows a few identified devices:

$ **lspci**

01:00.0 VGA compatible controller: NVIDIA Corporation GM107 [GeForce GTX 750 Ti] (rev a2)

04:02.0 Network controller: Ralink corp. RT2561/RT61 802.11g PCI

04:04.0 Multimedia audio controller: VIA Technologies Inc. ICE1712 [Envy24] PCI Multi-Channel I/O Controller (rev 02)

04:0b.0 FireWire (IEEE 1394): LSI Corporation FW322/323 [TrueFire] 1394a Controller (rev 70)

The output of such commands can be tens of lines long, so the previous and next examples contain only the sections of interest. The hexadecimal numbers at the beginning of each line are the unique addresses of the corresponding PCI device. The command lspci shows more details about a specific device if its address is given with option -s, accompanied by the option -v:

$ **lspci -s 04:02.0 -v**

04:02.0 Network controller: Ralink corp. RT2561/RT61 802.11g PCI

Subsystem: Linksys WMP54G v4.1

Flags: bus master, slow devsel, latency 32, IRQ 21

Memory at e3100000 (32-bit, non-prefetchable) [size=32K]

Capabilities: [40] Power Management version 2

kernel driver in use: rt61pci

The output now shows many more details of the device on address 04:02.0. It is a network controller, whose internal name is Ralink corp. RT2561/RT61 802.11g PCI. Subsystem is associated with the device’s brand and model — Linksys WMP54G v4.1 — and can be helpful for diagnostic purposes.

The kernel module can be identified in the line kernel driver in use, which shows the module rt61pci. From all the gathered information, it is correct to assume that:

The device was identified.

A matching kernel module was loaded.

The device should be ready for use.

Another way to verify which kernel module is in use for the specified device is provided by the option -k, available in more recent versions of lspci:

$ **lspci -s 01:00.0 -k**

01:00.0 VGA compatible controller: NVIDIA Corporation GM107 [GeForce GTX 750 Ti] (rev a2)

kernel driver in use: nvidia

kernel modules: nouveau, nvidia\_drm, nvidia

For the chosen device, an NVIDIA GPU board, lspci tells that the module in use is named nvidia, at line kernel driver in use: nvidia and all corresponding kernel modules are listed in the line kernel modules: nouveau, nvidia\_drm, nvidia.

Command lsusb is similar to lspci, but lists USB information exclusively:

$ **lsusb**

Bus 001 Device 029: ID 1781:0c9f Multiple Vendors USBtiny

Bus 001 Device 028: ID 093a:2521 Pixart Imaging, Inc. Optical Mouse

Bus 001 Device 020: ID 1131:1001 Integrated System Solution Corp. KY-BT100 Bluetooth Adapter

Bus 001 Device 011: ID 04f2:0402 Chicony Electronics Co., Ltd Genius LuxeMate i200 Keyboard

Bus 001 Device 007: ID 0424:7800 Standard Microsystems Corp.

Bus 001 Device 003: ID 0424:2514 Standard Microsystems Corp. USB 2.0 Hub

Bus 001 Device 002: ID 0424:2514 Standard Microsystems Corp. USB 2.0 Hub

Bus 001 Device 001: ID 1d6b:0002 Linux Foundation 2.0 root hub

Command lsusb shows the available USB channels and the devices connected to them. As with lspci, option -v displays more detailed output. A specific device can be selected for inspection by providing its ID to the option -d:

$ **lsusb -v -d 1781:0c9f**

Bus 001 Device 029: ID 1781:0c9f Multiple Vendors USBtiny

Device Descriptor:

bLength 18

bDescriptorType 1

bcdUSB 1.01

bDeviceClass 255 Vendor Specific Class

bDeviceSubClass 0

bDeviceProtocol 0

bMaxPacketSize0 8

idVendor 0x1781 Multiple Vendors

idProduct 0x0c9f USBtiny

bcdDevice 1.04

iManufacturer 0

iProduct 2 USBtiny

iSerial 0

bNumConfigurations 1

With option -t, command lsusb shows the current USB device mappings as a hierarchical tree:

$ **lsusb -t**

/: Bus 01.Port 1: Dev 1, Class=root\_hub, Driver=dwc\_otg/1p, 480M

|\_\_ Port 1: Dev 2, If 0, Class=Hub, Driver=hub/4p, 480M

|\_\_ Port 1: Dev 3, If 0, Class=Hub, Driver=hub/3p, 480M

|\_\_ Port 2: Dev 11, If 1, Class=Human Interface Device, Driver=usbhid, 1.5M

|\_\_ Port 2: Dev 11, If 0, Class=Human Interface Device, Driver=usbhid, 1.5M

|\_\_ Port 3: Dev 20, If 0, Class=Wireless, Driver=btusb, 12M

|\_\_ Port 3: Dev 20, If 1, Class=Wireless, Driver=btusb, 12M

|\_\_ Port 3: Dev 20, If 2, Class=Application Specific Interface, Driver=, 12M

|\_\_ Port 1: Dev 7, If 0, Class=Vendor Specific Class, Driver=lan78xx, 480M

|\_\_ Port 2: Dev 28, If 0, Class=Human Interface Device, Driver=usbhid, 1.5M

|\_\_ Port 3: Dev 29, If 0, Class=Vendor Specific Class, Driver=, 1.5M

It is possible that not all devices have corresponding modules associated with them. The communication with certain devices can be handled by the application directly, without the intermediation provided by a module. Nevertheless, there is significant information in the output provided by lsusb -t. When a matching module exists, its name appears at the end of the line for the device, as in Driver=btusb. The device Class identifies the general category, like Human Interface Device, Wireless, Vendor Specific Class, Mass Storage, among others. To verify which device is using the module btusb, present in the previous listing, both Bus and Dev numbers should be given to the -s option of the lsusb command:

$ **lsusb -s 01:20**

Bus 001 Device 020: ID 1131:1001 Integrated System Solution Corp. KY-BT100 Bluetooth Adapter

It is common to have a large set of loaded kernel modules in a standard Linux system at any time. The preferable way to interact with them is to use the commands provided by the kmod package, which is a set of tools to handle common tasks with Linux kernel modules like insert, remove, list, check properties, resolve dependencies and aliases. The lsmod command, for example, shows all currently loaded modules:

$ **lsmod**

Module Size Used by

kvm\_intel 138528 0

kvm 421021 1 kvm\_intel

iTCO\_wdt 13480 0

iTCO\_vendor\_support 13419 1 iTCO\_wdt

snd\_usb\_audio 149112 2

snd\_hda\_codec\_realtek 51465 1

snd\_ice1712 75006 3

snd\_hda\_intel 44075 7

arc4 12608 2

snd\_cs8427 13978 1 snd\_ice1712

snd\_i2c 13828 2 snd\_ice1712,snd\_cs8427

snd\_ice17xx\_ak4xxx 13128 1 snd\_ice1712

snd\_ak4xxx\_adda 18487 2 snd\_ice1712,snd\_ice17xx\_ak4xxx

microcode 23527 0

snd\_usbmidi\_lib 24845 1 snd\_usb\_audio

gspca\_pac7302 17481 0

gspca\_main 36226 1 gspca\_pac7302

videodev 132348 2 gspca\_main,gspca\_pac7302

rt61pci 32326 0

rt2x00pci 13083 1 rt61pci

media 20840 1 videodev

rt2x00mmio 13322 1 rt61pci

hid\_dr 12776 0

snd\_mpu401\_uart 13992 1 snd\_ice1712

rt2x00lib 67108 3 rt61pci,rt2x00pci,rt2x00mmio

snd\_rawmidi 29394 2 snd\_usbmidi\_lib,snd\_mpu401\_uart

The output of command lsmod is divided into three columns:

Module

Module name.

Size

Amount of RAM occupied by the module, in bytes.

Used by

Depending modules.

Some modules require other modules to work properly, as is the case with modules for audio devices:

$ **lsmod | fgrep -i snd\_hda\_intel**

snd\_hda\_intel 42658 5

snd\_hda\_codec 155748 3 snd\_hda\_codec\_hdmi,snd\_hda\_codec\_via,snd\_hda\_intel

snd\_pcm 81999 3 snd\_hda\_codec\_hdmi,snd\_hda\_codec,snd\_hda\_intel

snd\_page\_alloc 13852 2 snd\_pcm,snd\_hda\_intel

snd 59132 19 snd\_hwdep,snd\_timer,snd\_hda\_codec\_hdmi,snd\_hda\_codec\_via,snd\_pcm,snd\_seq,snd\_hda\_codec,snd\_hda\_intel,snd\_seq\_device

The third column, Used by, shows the modules that require the module in the first column to properly work. Many modules of the Linux sound architecture, prefixed by snd, are interdependent. When looking for problems during system diagnostics, it may be useful to unload specific modules currently loaded. Command modprobe can be used to both load and to unload kernel modules: to unload a module and its related modules, as long as they are not being used by a running process, command modprobe -r should be used. For example, to unload module snd-hda-intel (the module for a HDA Intel audio device) and other modules related to the sound system:

# **modprobe -r snd-hda-intel**

In addition to loading and unloading kernel modules while the system is running, it is possible to change module parameters when the kernel is being loaded, not so different from passing options to commands. Each module accepts specific parameters, but most times the default values are recommended and extra parameters are not needed. However, in some cases it is necessary to use parameters to change the behaviour of a module to work as expected.

Using the module name as the only argument, command modinfo shows a description, the file, the author, the license, the identification, the dependencies and the available parameters for the given module. Customized parameters for a module can be made persistent by including them in the file /etc/modprobe.conf or in individual files with the extension .conf in the directory /etc/modprobe.d/. Option -p will make command modinfo display all available parameters and ignore the other information:

# **modinfo -p nouveau**

vram\_pushbuf:Create DMA push buffers in VRAM (int)

tv\_norm:Default TV norm.

Supported: PAL, PAL-M, PAL-N, PAL-Nc, NTSC-M, NTSC-J,

hd480i, hd480p, hd576i, hd576p, hd720p, hd1080i.

Default: PAL

**NOTE** Ignored for cards with external TV encoders. (charp)

nofbaccel:Disable fbcon acceleration (int)

fbcon\_bpp:fbcon bits-per-pixel (default: auto) (int)

mst:Enable DisplayPort multi-stream (default: enabled) (int)

tv\_disable:Disable TV-out detection (int)

ignorelid:Ignore ACPI lid status (int)

duallink:Allow dual-link TMDS (default: enabled) (int)

hdmimhz:Force a maximum HDMI pixel clock (in MHz) (int)

config:option string to pass to driver core (charp)

debug:debug string to pass to driver core (charp)

noaccel:disable kernel/abi16 acceleration (int)

modeset:enable driver (default: auto, 0 = disabled, 1 = enabled, 2 = headless) (int)

atomic:Expose atomic ioctl (default: disabled) (int)

runpm:disable (0), force enable (1), optimus only default (-1) (int)

The sample output shows all the parameters available for module nouveau, a kernel module provided by the Nouveau Project as an alternative to the proprietary drivers for NVIDIA GPU cards. Option modeset, for example, allows to control whether display resolution and depth will be set in the kernel space rather than user space. Adding options nouveau modeset=0 to the file /etc/modprobe.d/nouveau.conf will disable the modeset kernel feature.

If a module is causing problems, the file /etc/modprobe.d/blacklist.conf can be used to block the loading of the module. For example, to prevent the automatic loading of the module nouveau, the line blacklist nouveau must be added to the file /etc/modprobe.d/blacklist.conf. This action is required when the proprietary module nvidia is installed and the default module nouveau should be set aside.

|  |  |
| --- | --- |
| **Note** | You can modify the /etc/modprobe.d/blacklist.conf file that already exists on the system by default. However, the preferred method is to create a separate configuration file, /etc/modprobe.d/<module\_name>.conf, that will contain settings specific only to the given kernel module. |

### Information Files and Device Files

The commands lspci, lsusb and lsmod act as front-ends to read hardware information stored by the operating system. This kind of information is kept in special files in the directories /proc and /sys. These directories are mount points to filesystems not present in a device partition, but only in RAM space used by the kernel to store runtime configuration and information on running processes. Such filesystems are not intended for conventional file storage, so they are called pseudo-filesystems and only exist while the system is running. The /proc directory contains files with information regarding running processes and hardware resources. Some of the important files in /proc for inspecting hardware are:

/proc/cpuinfo

Lists detailed information about the CPU(s) found by the operating system.

/proc/interrupts

A list of numbers of the interrupts per IO device for each CPU.

/proc/ioports

Lists currently registered Input/Output port regions in use.

/proc/dma

Lists the registered DMA (direct memory access) channels in use.

Files inside the /sys directory have similar roles to those in /proc. However, the /sys directory has the specific purpose of storing device information and kernel data related to hardware, whilst /proc also contains information about various kernel data structures, including running processes and configuration.

Another directory directly related to devices in a standard Linux system is /dev. Every file inside /dev is associated with a system device, particularly storage devices. A legacy IDE hard drive, for example, when connected to the motherboard’s first IDE channel, is represented by the file /dev/hda. Every partition in this disk will be identified by /dev/hda1, /dev/hda2 up to the last partition found.

Removable devices are handled by the udev subsystem, which creates the corresponding devices in /dev. The Linux kernel captures the hardware detection event and passes it to the udev process, which then identifies the device and dynamically creates corresponding files in /dev, using pre-defined rules.

In current Linux distributions, udev is responsible for the identification and configuration of the devices already present during machine power-up (coldplug detection) and the devices identified while the system is running (hotplug detection). Udev relies on SysFS, the pseudo filesystem for hardware related information mounted in /sys.

|  |  |
| --- | --- |
| **Note** | Hotplug is the term used to refer to the detection and configuration of a device while the system is running, such as when a USB device is inserted. The Linux kernel has supported hotplug features since version 2.6, allowing most system buses (PCI, USB, etc.) to trigger hotplug events when a device is connected or disconnected. |

As new devices are detected, udev searches for a matching rule in the pre-defined rules stored in the directory /etc/udev/rules.d/. The most important rules are provided by the distribution, but new ones can be added for specific cases.

### Storage Devices

In Linux, storage devices are generically referred as block devices, because data is read to and from these devices in blocks of buffered data with different sizes and positions. Every block device is identified by a file in the /dev directory, with the name of the file depending on the device type (IDE, SATA, SCSI, etc.) and its partitions. CD/DVD and floppy devices, for example, will have their names given accordingly in /dev: a CD/DVD drive connected to the second IDE channel will be identified as /dev/hdc (/dev/hda and /dev/hdb are reserved for the master and slave devices on the first IDE channel) and an old floppy drive will be identified as /dev/fdO, /dev/fd1, etc.

From Linux kernel version 2.4 onwards, most storage devices are now identified as if they were SCSI devices, regardless of their hardware type. IDE, SSD and USB block devices will be prefixed by sd. For IDE disks, the sd prefix will be used, but the third letter will be chosen depending on whether the drive is a master or slave (in the first IDE channel, master will be sda and slave will be sdb). Partitions are listed numerically. Paths /dev/sda1, /dev/sda2, etc. are used for the first and second partitions of the block device identified first and /dev/sdb1, /dev/sdb2, etc. used to identify the first and second partitions of the block device identified second. The exception to this pattern occurs with memory cards (SD cards) and NVMe devices (SSD connected to the PCI Express bus). For SD cards, the paths /dev/mmcblk0p1, /dev/mmcblk0p2, etc. are used for the first and second partitions of the device identified first and /dev/mmcblk1p1, /dev/mmcblk1p2, etc. used to identify the first and second partitions of the device identified second. NVMe devices receive the prefix nvme, as in /dev/nvme0n1p1 and /dev/nvme0n1p2.

## Guided Exercises

Suppose an operating system is unable to boot after adding a second SATA disk to the system. Knowing that all parts are not defective, what could be the possible cause for this error?

|  |
| --- |
|  |

Suppose you want to make sure the external video card connected to the PCI bus of your newly acquired desktop computer really is the one advertised by the manufacturer, but opening the computer case will void the warranty. What command could be used to list the details of the video card, as they were detected by the operating system?

|  |
| --- |
|  |

The following line is part of the output generated by command lspci:

03:00.0 RAID bus controller: LSI Logic / Symbios Logic MegaRAID SAS 2208 [Thunderbolt] (rev 05)

What command should you execute to identify the kernel module in use for this specific device?

|  |
| --- |
|  |

A system administrator wants to try different parameters for the bluetooth kernel module without rebooting the system. However, any attempt to unload the module with modprobe -r bluetooth results in the following error:

modprobe: FATAL: Module bluetooth is in use.

What is the possible cause for this error?

|  |
| --- |
|  |

## Explorational Exercises

It is not uncommon to find legacy machines in production environments, like when some equipment uses an outdated connection to communicate with the controlling computer, making it necessary to pay special attention to some peculiarities of these older machines. Some x86 servers with older BIOS firmware, for example, will not boot if a keyboard is not detected. How can this particular issue be avoided?

|  |
| --- |
|  |

Operating systems built around the Linux kernel are also available for a wide variety of computer architectures other than x86, like in single board computers based on the ARM architecture. An attentive user will notice the absence of the lspci command on such machines, like the Raspberry Pi. What difference with x86 machines justifies that absence?

|  |
| --- |
|  |

Many network routers have a USB port allowing for the connections of an external device, like a USB hard drive. Since most of these are using a Linux based operating system, how will an external USB hard drive be named in the /dev/ directory, assuming no other conventional block device is present in the router?

|  |
| --- |
|  |

In 2018, the hardware vulnerability known as Meltdown was discovered. It affects almost all processors of many architectures. Recent versions of the Linux kernel can inform if the current system is vulnerable. How can this information be obtained?

|  |
| --- |
|  |

## Summary

This lesson covers the general concepts on how the Linux kernel deals with hardware resources, mainly in the x86 architecture. The lesson goes through the following topics:

How settings defined in BIOS or UEFI configuration utilities can affect the way the operating system interacts with hardware.

How to use the tools provided by a standard Linux system to obtain information about the hardware.

How to identify permanent and removable storage devices in the filesystem.

The commands and procedures addressed were:

Commands to inspect detected hardware: lspci and lsusb.

Commands to manage kernel modules: lsmod and modprobe.

Hardware related special files, either the files found in the /dev/ directory or in the pseudo-filesystems in /proc/ and /sys/.

## Answers to Guided Exercises

Suppose an operating system is unable to boot after adding a second SATA disk to the system. Knowing that all parts are not defective, what could be the possible cause for this error?

The boot device order should be configured in the BIOS setup utility, otherwise the BIOS may not able to run the bootloader.

Suppose you want to make sure the external video card connected to the PCI bus of your newly acquired desktop computer really is the one advertised by the manufacturer, but opening the computer case will void the warranty. What command could be used to list the details of the video card, as they were detected by the operating system?

Command lspci will list detailed information about all devices currently connected to the PCI bus.

The following line is part of the output generated by command lspci:

03:00.0 RAID bus controller: LSI Logic / Symbios Logic MegaRAID SAS 2208 [Thunderbolt] (rev 05)

What command should you execute to identify the kernel module in use for this specific device?

The command lspci -s 03:00.0 -v or lspci -s 03:00.0 -k

A system administrator wants to try different parameters for the bluetooth kernel module without rebooting the system. However, any attempt to unload the module with modprobe -r bluetooth results in the following error:

modprobe: FATAL: Module bluetooth is in use.

What is the possible cause for this error?

Module bluetooth is being used by a running process.

## Answers to Explorational Exercises

It is not uncommon to find legacy machines in production environments, like when some equipment uses an outdated connection to communicate with the controlling computer, making it necessary to pay special attention to some peculiarities of these older machines. Some x86 servers with older BIOS firmwares, for example, will not boot if a keyboard is not detected. How this particular issue can be avoided?

The BIOS configuration utility has an option to deactivate the locking of the computer when a keyboard is not found.

Operating systems built around the Linux kernel are also available for a wide variety of computer architectures other than x86, like in single board computers based on the ARM architecture. An attentive user will notice the absence of the lspci command on such machines, like the Raspberry Pi. What difference with x86 machines justifies that absence?

Unlike most x86 machines, an ARM based computer like the Raspberry Pi lacks a PCI bus, so the command lspci is useless.

Many network routers have a USB port allowing for the connections of an external device, like a USB hard drive. Since most of these are using a Linux based operating system, how will an external USB hard drive be named in the /dev/ directory, assuming no other conventional block device is present in the router?

Modern Linux kernels identify USB hard drives as SATA devices, so the corresponding file will be /dev/sda as no other conventional block device exists in the system.

In 2018, the hardware vulnerability known as Meltdown was discovered. It affects almost all processors of many architectures. Recent versions of the Linux kernel can inform if the current system is vulnerable. How can this information be obtained?

The file /proc/cpuinfo has a line showing the known bugs for the corresponding CPU, like bugs: cpu\_meltdown.

© 2020 Linux Professional Insitute Inc. All rights reserved. Visit the Learning Materials website: https://learning.lpi.org  
This work is licensed under the Creative Commons Attribution-NonCommercial-NoDerivatives 4.0 International License.