BIOS_or_UEFI

(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.

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.

Introduction
In order to control the machine, the operating system’s main component — the kernel — must be loaded by a program called a bootloader, which itself is loaded by a pre-installed firmware such as BIOS or UEFI. The bootloader can be customized to pass parameters to the kernel, such as which partition contains the root filesystem or in which mode the operating system should execute. Once loaded the kernel continues the boot process identifying and configuring the hardware. Lastly, the kernel calls the utility responsible for starting and managing the system’s services.

Note
On some Linux distributions, commands executed in this lesson may require root privileges.

BIOS or UEFI
The procedures executed by x86 machines to run the bootloader are different whether they use BIOS or UEFI. The BIOS, short for Basic Input/Output System, is a program stored in a non-volatile memory chip attached to the motherboard, executed every time the computer is powered on. This type of program is called firmware and its storage location is separate from the other storage devices the system may have. The BIOS assumes that the first 440 bytes in the first storage device — following the order defined in the BIOS configuration utility — are the first stage of the bootloader (also called bootstrap). The first 512 bytes of a storage device are named the MBR (Master Boot Record) of storage devices using the standard DOS partition schema and, in addition to the first stage of the bootloader, contain the partition table. If the MBR does not contain the correct data, the system will not be able to boot, unless an alternative method is employed.

Generally speaking, the pre-operating steps to boot a system equipped with BIOS are:

The POST (power-on self-test) process is executed to identify simple hardware failures as soon as the machine is powered on.

The BIOS activates the basic components to load the system, like video output, keyboard and storage media.

The BIOS loads the first stage of the bootloader from the MBR (the first 440 bytes of the first device, as defined in the BIOS configuration utility).

The first stage of the bootloader calls the second stage of the bootloader, responsible for presenting boot options and loading the kernel.

The UEFI, short for Unified Extensible Firmware Interface, differs from BIOS in some key points. As the BIOS, the UEFI is also a firmware, but it can identify partitions and read many filesystems found in them. The UEFI does not rely on the MBR, taking into account only the settings stored in its non-volatile memory (NVRAM) attached to the motherboard. These definitions indicate the location of the UEFI compatible programs, called EFI applications, that will be executed automatically or called from a boot menu. EFI applications can be bootloaders, operating system selectors, tools for system diagnostics and repair, etc. They must be in a conventional storage device partition and in a compatible filesystem. The standard compatible filesystems are FAT12, FAT16 and FAT32 for block devices and ISO-9660 for optical media. This approach allows for the implementation of much more sophisticated tools than those possible with BIOS.

The partition containing the EFI applications is called the EFI System Partition or just ESP. This partition must not be shared with other system filesystems, like the root filesystem or user data filesystems. The EFI directory in the ESP partition contains the applications pointed to by the entries saved in the NVRAM.

Generally speaking, the pre-operating system boot steps on a system with UEFI are:

The POST (power-on self-test) process is executed to identify simple hardware failures as soon as the machine is powered on.

The UEFI activates the basic components to load the system, like video output, keyboard and storage media.

UEFI's firmware reads the definitions stored in NVRAM to execute the pre-defined EFI application stored in the ESP partition's filesystem. Usually, the pre-defined EFI application is a bootloader.

If the pre-defined EFI application is a bootloader, it will load the kernel to start the operating system.

The UEFI standard also supports a feature called Secure Boot, which only allows the execution of signed EFI applications, that is, EFI applications authorized by the hardware manufacturer. This feature increases the protection against malicious software, but can make it difficult to install operating systems not covered by the manufacturer's warranty.

Commands_for_Inspection

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.

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.

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.

lsusb

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.

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
```

lsmod

lsmod
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.

modprobe

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.

modinfo

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.

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.

/sys

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.

/dev

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.

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.

The_Bootloader

The most popular bootloader for Linux in the x86 architecture is GRUB (Grand Unified Bootloader). As soon as it is called by the BIOS or by the UEFI, GRUB displays a list of operating systems available to boot. Sometimes the list does not appear automatically, but it can be invoked by pressing Shift while GRUB is being called by BIOS. In UEFI systems, the Esc key should be used instead.

From the GRUB menu it is possible to choose which one of the installed kernels should be loaded and to pass new parameters to it. Most kernel parameters follow the pattern option=value. Some of the most useful kernel parameters are:

acpi
Enables/disables ACPI support. acpi=off will disable support for ACPI.

init
Sets an alternative system initiator. For example, init=/bin/bash will set the Bash shell as the initiator. This means that a shell session will start just after the kernel boot process.

systemd.unit
Sets the systemd target to be activated. For example, systemd.unit=graphical.target. Systemd also accepts the numerical runlevels as defined for SysV. To activate the runlevel 1, for example, it is only necessary to include the number 1 or the letter S (short for “single”) as a kernel parameter.

mem
Sets the amount of available RAM for the system. This parameter is useful for virtual machines so as to limit how much RAM will be available to each guest. Using mem=512M will limit to 512 megabytes the amount of available RAM to a particular guest system.

System_Initialization

Apart from the kernel, the operating system depends on other components that provide the expected features. Many of these components are loaded during the system initialization process, varying from simple shell scripts to more complex service programs. Scripts are often used to perform short lived tasks that will run and terminate during the system initialization process. Services, also known as daemons, may be active all the time as they can be responsible for intrinsic aspects of the operating system.

The diversity of ways that startup scripts and daemons with the most different characteristics can be built into a Linux distribution is huge, a fact that historically hindered the development of a single solution that meets the expectations of maintainers and users of all Linux distributions. However, any tool that the distribution maintainers have chosen to perform this function will at least be able to start, stop and restart system services. These actions are often performed by the system itself after a software update, for example, but the system administrator will almost always need to manually restart the service after making modifications to its configuration file.

It is also convenient for a system administrator to be able to activate a particular set of daemons, depending on the circumstances. It should be possible, for example, to run just a minimum set of services in order to perform system maintenance tasks.

Note
Strictly speaking, the operating system is just the kernel and its components which control the hardware and manages all processes. It is common, however, to use the term “operating system” more loosely, to designate an entire group of distinct programs that make up the software environment where the user can perform the basic computational tasks.

The initialization of the operating system starts when the bootloader loads the kernel into RAM. Then, the kernel will take charge of the CPU and will start to detect and setup the fundamental aspects of the operating system, like basic hardware configuration and memory addressing.

The kernel will then open the initramfs (initial RAM filesystem). The initramfs is an archive containing a filesystem used as a temporary root filesystem during the boot process. The main purpose of an initramfs file is to provide the required modules so the kernel can access the “real” root filesystem of the operating system.

init

As soon as the root filesystem is available, the kernel will mount all filesystems configured in /etc/fstab and then will execute the first program, a utility named init. The init program is responsible for running all initialization scripts and system daemons. There are distinct implementations of such system initiators apart from the traditional init, like systemd and Upstart. Once the init program is loaded, the initramfs is removed from RAM.

SysV_standard

A service manager based on the SysVinit standard controls which daemons and resources will be available by employing the concept of runlevels. Runlevels are numbered 0 to 6 and are designed by the distribution maintainers to fulfill specific purposes. The only runlevel definitions shared between all distributions are the runlevels 0, 1 and 6.

A common feature among operating systems following Unix design principles is the employment of separate processes to control distinct functions of the system. These processes, called daemons (or, more generally, services), are also responsible for extended features underlying the operating system, like network application services (HTTP server, file sharing, email, etc.), databases, on-demand configuration, etc. Although Linux utilizes a monolithic kernel, many low level aspects of the operating system are affected by daemons, like load balancing and firewall configuration.

Which daemons should be active depends on the purpose of the system. The set of active daemons should also be modifiable at runtime, so services can be started and stopped without having to reboot the whole system. To tackle this issue, every major Linux distribution offers some form of service management utility to control the system.

Services can be controlled by shell scripts or by a program and its supporting configuration files. The first method is implemented by the SysVinit standard, also known as System V or just SysV. The second method is implemented by systemd and Upstart. Historically, SysV based service managers were the most used by Linux distributions. Today, systemd based service managers are more often found in most Linux distributions. The service manager is the first program launched by the kernel during the boot process, so its PID (process identification number) is always 1.

SysVinit
A service manager based on the SysVinit standard will provide predefined sets of system states, called runlevels, and their corresponding service script files to be executed. Runlevels are numbered 0 to 6, being generally assigned to the following purposes:

Runlevel 0
System shutdown.

Runlevel 1, s or single
Single user mode, without network and other non-essential capabilities (maintenance mode).

Runlevel 2, 3 or 4
Multi-user mode. Users can log in by console or network. Runlevels 2 and 4 are not often used.

Runlevel 5
Multi-user mode. It is equivalent to 3, plus the graphical mode login.

Runlevel 6
System restart.

The program responsible for managing runlevels and associated daemons/resources is /sbin/init. During system initialization, the init program identifies the requested runlevel, defined by a kernel parameter or in the /etc/inittab file, and loads the associated scripts listed there for the given runlevel. Every runlevel may have many associated service files, usually scripts in the /etc/init.d/ directory. As not all runlevels are equivalent through different Linux distributions, a short description of the runlevel's purpose can also be found in SysV based distributions.

The syntax of the /etc/inittab file uses this format:

id:runlevels:action:process
The id is a generic name up to four characters in length used to identify the entry. The runlevels entry is a list of runlevel numbers for which a specified action should be executed. The action term defines how init will execute the process indicated by the term process. The available actions are:

boot
The process will be executed during system initialization. The field runlevels is ignored.

bootwait
The process will be executed during system initialization and init will wait until it finishes to continue. The field runlevels is ignored.

sysinit
The process will be executed after system initialization, regardless of runlevel. The field runlevels is ignored.

wait
The process will be executed for the given runlevels and init will wait until it finishes to continue.

respawn
The process will be restarted if it is terminated.

ctrlaltdel
The process will be executed when the init process receives the SIGINT signal, triggered when the key sequence of Ctrl+Alt+Del is pressed.

The default runlevel — the one that will be chosen if no other is given as a kernel parameter — is also defined in /etc/inittab, in the entry id:x:initdefault. The x is the number of the default runlevel. This number should never be 0 or 6, given that it would cause the system to shutdown or restart as soon as it finishes the boot process. A typical /etc/inittab file is shown below:

```
# Default runlevel
id:3:initdefault:

# Configuration script executed during boot
si::sysinit:/etc/init.d/rcS

# Action taken on runlevel S (single user)
~:S:wait:/sbin/sulogin

# Configuration for each execution level
l0:0:wait:/etc/init.d/rc 0
l1:1:wait:/etc/init.d/rc 1
l2:2:wait:/etc/init.d/rc 2
l3:3:wait:/etc/init.d/rc 3
l4:4:wait:/etc/init.d/rc 4
l5:5:wait:/etc/init.d/rc 5
l6:6:wait:/etc/init.d/rc 6

# Action taken upon ctrl+alt+del keystroke
ca::ctrlaltdel:/sbin/shutdown -r now
```

```
# Enable consoles for runlevels 2 and 3
1:23:respawn:/sbin/getty tty1 VC linux
2:23:respawn:/sbin/getty tty2 VC linux
3:23:respawn:/sbin/getty tty3 VC linux
4:23:respawn:/sbin/getty tty4 VC linux

# For runlevel 3, also enable serial
# terminals ttyS0 and ttyS1 (modem) consoles
S0:3:respawn:/sbin/getty -L 9600 ttyS0 vt320
S1:3:respawn:/sbin/mgetty -x0 -D ttyS1
```

The telinit q command should be executed every time after the /etc/inittab file is modified. The argument q (or Q) tells init to reload its configuration. Such a step is important to avoid a system halt due to an incorrect configuration in /etc/inittab.

The scripts used by init to setup each runlevel are stored in the directory /etc/init.d/. Every runlevel has an associated directory in /etc/, named /etc/rc0.d/, /etc/rc1.d/, /etc/rc2.d/, etc., with the scripts that should be executed when the corresponding runlevel starts. As the same script can be used by different runlevels, the files in those directories are just symbolic links to the actual scripts in /etc/init.d/. Furthermore, the first letter of the link filename in the runlevel's directory indicates if the service should be started or terminated for the corresponding runlevel. A link's filename starting with letter K determines that the service will be killed when entering the runlevel (kill). Starting with letter S, the service will be started when entering the runlevel (start). The directory /etc/rc1.d/, for example, will have many links to network scripts beginning with letter K, considering that the runlevel 1 is the single user runlevel, without network connectivity.

The command runlevel shows the current runlevel for the system. The runlevel command shows two values, the first is the previous runlevel and the second is the current runlevel:

```
$ runlevel
N 3
```

The letter N in the output shows that the runlevel has not changed since last boot. In the example, the runlevel 3 is the current runlevel of the system.

The same init program can be used to alternate between runlevels in a running system, without the need to reboot. The command telinit can also be used to alternate between runlevels. For example, commands telinit 1, telinit s or telinit S will change the system to runlevel 1.

systemd

systemd is a modern system and services manager with a compatibility layer for the SysV commands and runlevels. systemd has a concurrent structure, employs sockets and D-Bus for service activation, on-demand daemon execution, process monitoring with cgroups, snapshot support, system session recovery, mount point control and a dependency-based service control. In recent years most major Linux distributions have gradually adopted systemd as their default system manager.

systemd
Currently, systemd is the most widely used set of tools to manage system resources and services, which are referred to as units by systemd. A unit consists of a name, a type and a corresponding configuration file. For example, the unit for a httpd server process (like the Apache web server) will be httpd.service on Red Hat based distributions and its configuration file will also be called httpd.service (on Debian based distributions this unit is named apache2.service).

There are seven distinct types of systemd units:

service
The most common unit type, for active system resources that can be initiated, interrupted and reloaded.

socket
The socket unit type can be a filesystem socket or a network socket. All socket units have a corresponding service unit, loaded when the socket receives a request.

device
A device unit is associated with a hardware device identified by the kernel. A device will only be taken as a systemd

unit if a udev rule for this purpose exists. A device unit can be used to resolve configuration dependencies when certain hardware is detected, given that properties from the udev rule can be used as parameters for the device unit.

mount
A mount unit is a mount point definition in the filesystem, similar to an entry in /etc/fstab.

automount
An automount unit is also a mount point definition in the filesystem, but mounted automatically. Every automount unit has a corresponding mount unit, which is initiated when the automount mount point is accessed.

target
A target unit is a grouping of other units, managed as a single unit.

snapshot
A snapshot unit is a saved state of the systemd manager (not available on every Linux distribution).

The main command for controlling systemd units is systemctl. Command systemctl is used to execute all tasks regarding unit activation, deactivation, execution, interruption, monitoring, etc. For a fictitious unit called unit.service, for example, the most common systemctl actions will be:

systemctl start unit.service
Starts unit.

systemctl stop unit.service
Stops unit.

systemctl restart unit.service
Restarts unit.

systemctl status unit.service
Shows the state of unit, including if it is running or not.

systemctl is-active unit.service
Shows active if unit is running or inactive otherwise.

systemctl enable unit.service
Enables unit, that is, unit will load during system initialization.

systemctl disable unit.service
unit will not start with the system.

systemctl is-enabled unit.service
Verifies if unit starts with the system. The answer is stored in the variable $?. The value 0 indicates that unit starts with the system and the value 1 indicates that unit does not starts with the system.

Note
Newer installations of systemd will actually list a unit’s configuration for boot time. For example:

$ systemctl is-enabled apparmor.service
enabled
If no other units with the same name exist in the system, then the suffix after the dot can be dropped. If, for example, there is only one httpd unit of type service, then only httpd is enough as the unit parameter for systemctl.

The systemctl command can also control system targets. The multi-user.target unit, for example, combines all units required by the multi-user system environment. It is similar to the runlevel number 3 in a system utilizing SysV.

Command systemctl isolate alternates between different targets. So, to manually alternate to target multi-user:

systemctl isolate multi-user.target
There are corresponding targets to SysV runlevels, starting with runlevelO.target up to runlevel6.target. However, systemd does not use the /etc/inittab file. To change the default system target, the option systemd.unit can be added to the kernel parameters list. For example, to use multi-user.target as the standard target, the kernel parameter should be systemd.unit=multi-user.target. All kernel parameters can be made persistent by changing the bootloader configuration.

Another way to change the default target is to modify the symbolic link /etc/systemd/system/default.target so it points to the desired target. The redefinition of the link can be done with the systemctl command by itself:

systemctl set-default multi-user.target
Likewise, you can determine what your system's default boot target is with the following command:

$ systemctl get-default
graphical.target
Similar to systems adopting SysV, the default target should never point to shutdown.target, as it corresponds to the runlevel 0 (shutdown).

The configuration files associated with every unit can be found in the /lib/systemd/system/ directory. The command systemctl list-unit-files lists all available units and shows if they are enabled to start when the system boots. The option --type will select only the units for a given type, as in systemctl list-unit-files --type=service and systemctl list-unit-files --type=target.

Active units or units that have been active during the current system session can be listed with command systemctl list-units. Like the list-unit-files option, the systemctl list-units --type=service command will select only units of type service and command systemctl list-units --type=target will select only units of type target.

systemd is also responsible for triggering and responding to power related events. The systemctl suspend command will put the system in low power mode, keeping current data in memory. Command systemctl hibernate will copy all memory data to disk, so the current state of the system can be recovered after powering it off. The actions associated with such events are defined in the file /etc/systemd/logind.conf or in separate files inside the directory /etc/systemd/logind.conf.d/. However, this systemd feature can only be used when there is no other power manager running in the system, like the acpid daemon. The acpid daemon is the main power manager for Linux and allows finer adjustments to the actions following power related events, like closing the laptop lid, low battery or battery charging levels.

Upstart

Like systemd, Upstart is a substitute to init. The focus of Upstart is to speed up the boot process by parallelizing the loading process of system services. Upstart was used by Ubuntu based distributions in past releases, but today gave way to systemd.

Upstart
The initialization scripts used by Upstart are located in the directory /etc/init/. System services can be listed with command initctl list, which also shows the current state of the services and, if available, their PID number.

initctl list
avahi-cups-reload stop/waiting
avahi-daemon start/running, process 1123
mountall-net stop/waiting
mountnfs-bootclean.sh start/running
nmbd start/running, process 3085
passwd stop/waiting
rc stop/waiting
rsyslog start/running, process 1095
tty4 start/running, process 1761
udev start/running, process 1073
upstart-udev-bridge start/running, process 1066
console-setup stop/waiting
irqbalance start/running, process 1842
plymouth-log stop/waiting
smbd start/running, process 1457
tty5 start/running, process 1764
failsafe stop/waiting
Every Upstart action has its own independent command. For example, command start can be used to initiate a sixth virtual terminal:

start tty6
The current state of a resource can be verified with command status:

```
# status tty6
tty6 start/running, process 3282
```
And the interruption of a service is done with the command stop:

```
# stop tty6
```
Upstart does not use the /etc/inittab file to define runlevels, but the legacy commands runlevel and telinit can still be used to verify and alternate between runlevels.

Initialization_Inspection

Errors may occur during the boot process, but they may not be so critical to completely halt the operating system. Notwithstanding, these errors may compromise the expected behaviour of the system. All errors result in messages that can be used for future investigations, as they contain valuable information about when and how the error occurred. Even when no error messages are generated, the information collected during the boot process can be useful for tuning and configuration purposes.

The memory space where the kernel stores its messages, including the boot messages, is called the kernel ring buffer. The messages are kept in the kernel ring buffer even when they are not displayed during the initialization process, like when an animation is displayed instead. However the kernel ring buffer loses all messages when the system is turned off or by executing the command dmesg --clear.

dmesg

Without options, command dmesg displays the current messages in the kernel ring buffer:

```
$ dmesg
[    5.262389] EXT4-fs (sda1): mounted filesystem with ordered data mode. Opts: (null)
[    5.449712] ip_tables: (C) 2000-2006 Netfilter Core Team
[    5.460286] systemd[1]: systemd 237 running in system mode.
[    5.480138] systemd[1]: Detected architecture x86-64.
[    5.481767] systemd[1]: Set hostname to <torre>.
[    5.636607] systemd[1]: Reached target User and Group Name Lookups.
[    5.636866] systemd[1]: Created slice System Slice.
[    5.637000] systemd[1]: Listening on Journal Audit Socket.
[    5.637085] systemd[1]: Listening on Journal Socket.
[    5.637827] systemd[1]: Mounting POSIX Message Queue File System...
[    5.638639] systemd[1]: Started Read required files in advance.
[    5.641661] systemd[1]: Starting Load Kernel Modules...
[    5.661672] EXT4-fs (sda1): re-mounted. Opts: errors=remount-ro
[    5.694322] lp: driver loaded but no devices found
[    5.702609] ppdev: user-space parallel port driver
[    5.705384] parport_pc 00:02: reported by Plug and Play ACPI
[    5.705468] parport0: PC-style at 0x378 (0x778), irq 7, dma 3 [PCSPP,TRISTATE,COMPAT,EPP,ECP,DMA]
[    5.800146] lp0: using parport0 (interrupt-driven).
[    5.897421] systemd-journald[352]: Received request to flush runtime journal from PID 1
```

The output of dmesg can be hundreds of lines long, so the previous listing contains only the excerpt showing the kernel calling the systemd service manager. The values in the beginning of the lines are the amount of seconds relative to when kernel load begins.

journalctl

In systems based on systemd, command journalctl will show the initialization messages with options -b, --boot, -k or --dmesg. Command journalctl --list-boots shows a list of boot numbers relative to the current boot, their identification hash and the timestamps of the first and last corresponding messages:

```
$ journalctl --list-boots
 -4 9e5b3eb4952845208b841ad4dbefa1a6 Thu 2019-10-03 13:39:23 -03—Thu 2019-10-03 13:40:30 -03
 -3 9e3d79955535430aa43baa17758f40fa Thu 2019-10-03 13:41:15 -03—Thu 2019-10-03 14:56:19 -03
 -2 17672d8851694e6c9bb102df7355452c Thu 2019-10-03 14:56:57 -03—Thu 2019-10-03 19:27:16 -03
```

-1 55c0d9439bfb4e85a20a62776d0dbb4d Thu 2019-10-03 19:27:53 -03—Fri 2019-10-04 00:28:47 -03
0 08fbbebd9f964a74b8a02bb27b200622 Fri 2019-10-04 00:31:01 -03—Fri 2019-10-04 10:17:01 -03
Previous initialization logs are also kept in systems based on systemd, so messages from prior operating system sessions can still be inspected. If options -b 0 or --boot=0 are provided, then messages for the current boot will be shown. Options -b -1 or --boot=-1 will show messages from the previous initialization. Options -b -2 or --boot=-2 will show the messages from the initialization before that and so on. The following excerpt shows the kernel calling the systemd service manager for the last initialization process:

$ journalctl -b 0
oct 04 00:31:01 ubuntu-host kernel: EXT4-fs (sda1): mounted filesystem with ordered data mode. Opts: (null)
oct 04 00:31:01 ubuntu-host kernel: ip_tables: (C) 2000-2006 Netfilter Core Team
oct 04 00:31:01 ubuntu-host systemd[1]: systemd 237 running in system mode.
oct 04 00:31:01 ubuntu-host systemd[1]: Detected architecture x86-64.
oct 04 00:31:01 ubuntu-host systemd[1]: Set hostname to <torre>.
oct 04 00:31:01 ubuntu-host systemd[1]: Reached target User and Group Name Lookups.
oct 04 00:31:01 ubuntu-host systemd[1]: Created slice System Slice.
oct 04 00:31:01 ubuntu-host systemd[1]: Listening on Journal Audit Socket.
oct 04 00:31:01 ubuntu-host systemd[1]: Listening on Journal Socket.
oct 04 00:31:01 ubuntu-host systemd[1]: Mounting POSIX Message Queue File System...
oct 04 00:31:01 ubuntu-host systemd[1]: Started Read required files in advance.
oct 04 00:31:01 ubuntu-host systemd[1]: Starting Load Kernel Modules...
oct 04 00:31:01 ubuntu-host kernel: EXT4-fs (sda1): re-mounted. Opts: commit=300,barrier=0,errors=remount-ro
oct 04 00:31:01 ubuntu-host kernel: lp: driver loaded but no devices found
oct 04 00:31:01 ubuntu-host kernel: ppdev: user-space parallel port driver
oct 04 00:31:01 ubuntu-host kernel: parport_pc 00:02: reported by Plug and Play ACPI
oct 04 00:31:01 ubuntu-host kernel: parport0: PC-style at 0x378 (0x778), irq 7, dma 3
[PCSPP,TRISTATE,COMPAT,EPP,ECP,DMA]
oct 04 00:31:01 ubuntu-host kernel: lp0: using parport0 (interrupt-driven).
oct 04 00:31:01 ubuntu-host systemd-journald[352]: Journal started
oct 04 00:31:01 ubuntu-host systemd-journald[352]: Runtime journal (/run/log/journal/
abb765408f3741ae9519ab3b96063a15) is 4.9M, max 39.4M, 34.5M free.
oct 04 00:31:01 ubuntu-host systemd-modules-load[335]: Inserted module 'lp'
oct 04 00:31:01 ubuntu-host systemd-modules-load[335]: Inserted module 'ppdev'
oct 04 00:31:01 ubuntu-host systemd-modules-load[335]: Inserted module 'parport_pc'
oct 04 00:31:01 ubuntu-host systemd[1]: Starting Flush Journal to Persistent Storage...

Shutdown_and_Restart

A very traditional command used to shutdown or restart the system is unsurprisingly called shutdown. The shutdown command adds extra functions to the power off process: it automatically notifies all logged-in users with a warning message in their shell sessions and new logins are prevented. Command shutdown acts as an intermediary to SysV or systemd procedures, that is, it executes the requested action by calling the corresponding action in the services manager adopted by the system.

After shutdown is executed, all processes receive the SIGTERM signal, followed by the SIGKILL signal, then the system shuts down or changes its runlevel. By default, when neither options -h or -r are used, the system alternates to runlevel 1, that is, the single user mode. To change the default options for shutdown, the command should be executed with the following syntax:

$ shutdown [option] time [message]
Only the parameter time is required. The time parameter defines when the requested action will be executed, accepting the following formats:

hh:mm
This format specifies the execution time as hour and minutes.

+m
This format specifies how many minutes to wait before execution.

now or +0
This format determines immediate execution.

The message parameter is the warning text sent to all terminal sessions of logged-in users.

The SysV implementation allows for the limiting of users that will be able to restart the machine by pressing

Ctrl+Alt+Del. This is possible by placing option -a for the shutdown command present at the line regarding ctrlaltdel in the /etc/inittab file. By doing this, only users whose usernames are in the /etc/shutdown.allow file will be able to restart the system with the Ctrl+Alt+Del keystroke combination.

The systemctl command can also be used to turn off or to restart the machine in systems employing systemd. To restart the system, the command systemctl reboot should be used. To turn off the system, the command systemctl poweroff should be used. Both commands require root privileges to run, as ordinary users can not perform such procedures.

Note
Some Linux distributions will link poweroff and reboot to systemctl as individual commands. For example:

```
$ sudo which poweroff
/usr/sbin/poweroff
$ sudo ls -l /usr/sbin/poweroff
lrwxrwxrwx 1 root root 14 Aug 20 07:50 /usr/sbin/poweroff -> /bin/systemctl
```

Not all maintenance activities require the system to be turned off or restarted. However, when it is necessary to change the system's state to single-user mode, it is important to warn logged-in users so that they are not harmed by an abrupt termination of their activities.

Similar to what the shutdown command does when powering off or restarting the system, the wall command is able to send a message to terminal sessions of all logged-in users. To do so, the system administrator only needs to provide a file or directly write the message as a parameter to command wall.

Design_hard_disk_layout

Introduction
To succeed in this objective, you need to understand the relationship between disks, partitions, filesystems and volumes.

Think of a disk (or storage device, since modern devices do not contain any "disks" at all) as a "physical container" for you data.

Before a disk can be used by a computer it needs to be partitioned. A partition is a logical subset of the physical disk, like a logical "fence". Partitioning is a way to "compartmentalize" information stored on the disk, separating, for example, operating system data from user data.

Every disk needs at least one partition, but can have multiple partitions if needed, and information about them is stored in a partition table. This table includes information about the first and last sectors of the partition and its type, as well as further details on each partition.

Inside each partition there is a filesystem. The filesystem describes the way the information is actually stored on the disk. This information includes how the directories are organized, what is the relationship between them, where is the data for each file, etc.

Partitions cannot span multiple disks. But using the Logical Volume Manager (LVM) multiple partitions can be combined, even across disks, to form a single logical volume.

Logical volumes abstract the limitations of the physical devices and let your work with "pools" of disk space that can be combined or distributed in a much more flexible way than traditional partitions. LVM is useful in situations where you would need to add more space to a partition without having to migrate the data to a larger device.

In this objective you will learn how to design a disk partitioning scheme for a Linux system, allocating filesystems and swap space to separate partitions or disks when needed.

How to create and manage partitions and filesystems will be discussed in other lessons. We will discuss an overview of LVM in this objective, but a detailed explanation is out of the scope.

Mount_Points

Before a filesystem can be accessed on Linux it needs to be mounted. This means attaching the filesystem to a

specific point in your system’s directory tree, called a mount point.

When mounted, the contents of the filesystem will be available under the mount point. For example, imagine you have a partition with your users' personal data (their home directories), containing the directories /john, /jack and /carol. When mounted under /home, the contents of those directories will be available under /home/john, /home/jack and /home/carol.

The mount point must exist before mounting the filesystem. You cannot mount a partition under /mnt/userdata if this directory does not exist. However if the directory does exist and contains files, those files will be unavailable until you unmount the filesystem. If you list the contents of the directory, you will see the files stored on the mounted filesystem, not the original contents of the directory.

Filesystems can be mounted anywhere you want. However, there are some good practices that should be followed to make system administration easier.

Traditionally, /mnt was the directory under which all external devices would be mounted and a number of pre-configured anchor points for common devices, like CD-ROM drives (/mnt/cdrom) and floppy disks (/mnt/floppy) existed under it.

This has been superseded by /media, which is now the default mount point for any user-removable media (e.g. external disks, USB flash drives, memory card readers, optical disks, etc.) connected to the system.

On most modern Linux distributions and desktop environments, removable devices are automatically mounted under /media/USER/LABEL when connected to the system, where USER is the username and LABEL is the device label. For example, a USB flash drive with the label FlashDrive connected by the user john would be mounted under /media/john/FlashDrive/. The way this is handled is different depending on the desktop environment.

That being said, whenever you need to manually mount a filesystem, it is good practice to mount it under /mnt. The specific commands to control the mounting and unmounting of filesystems under Linux will be discussed in another lesson.

The_Boot_Partition(/boot)

Keeping Things Separated
On Linux, there are some directories that you should consider keeping on separate partitions. There are many reasons for this: for example, by keeping bootloader-related files (stored on /boot) on a boot partition, you ensure your system will still be able to boot in case of a crash on the root filesystem.

Keeping user’s personal directories (under /home) on a separate partition makes it easier to reinstall the system without the risk of accidentally touching user data. Keeping data related to a web or database server (usually under /var) on a separate partition (or even a separate disk) makes system administration easier should you need to add more disk space for those use cases.

There may even be performance reasons to keep certain directories on separate partitions. You may want to keep the root filesystem (/) on a speedy SSD unit, and bigger directories like /home and /var on slower hard disks which offer much more space for a fraction of the cost.

The Boot Partition (/boot)
The boot partition contains files used by the bootloader to load the operating system. On Linux systems the bootloader is usually GRUB2 or, on older systems, GRUB Legacy. The partition is usually mounted under /boot and its files are stored in /boot/grub.

Technically a boot partition is not needed, since in most cases GRUB can mount the root partition (/) and load the files from a separate /boot directory.

However, a separate boot partition may be desired for safety (ensuring the system will boot even in case of a root filesystem crash), or if you wish to use a filesystem which the bootloader cannot understand in the root partition, or if it uses an unsupported encryption or compression method.

The boot partition is usually the first partition on the disk. This is because the original IBM PC BIOS addressed disks using cylinders, heads and sectors (CHS), with a maximum of 1024 cylinders, 256 heads and 63 sectors, resulting in a maximum disk size of 528 MB (504 MB under MS-DOS). This means that anything past this mark would not be accessible on legacy systems, unless a different disk addressing scheme (like Logical Block Addressing, LBA) was

used.

So for maximum compatibility, the boot partition is usually located at the start of the disk and ends before cylinder 1024 (528 MB), ensuring that no matter what, the machine will be always able to load the kernel.

Since the boot partition only stores the files needed by the bootloader, the initial RAM disk and kernel images, it can be quite small by today's standards. A good size is around 300 MB.

The_EFI_System_Partition(ESP)

The EFI System Partition (ESP) is used by machines based on the Unified Extensible Firmware Interface (UEFI) to store boot loaders and kernel images for the operating systems installed.

This partition is formatted in a FAT-based filesystem. On a disk partitioned with a GUID Partition Table it has a globally unique identifier of C12A7328-F81F-11D2-BA4B-00A0C93EC93B. If the disk was formatted under the MBR partitioning scheme the partition ID is 0xEF.

On machines running Microsoft Windows this partition is usually the first one on the disk, although this is not required. The ESP is created (or populated) by the operating system upon installation, and on a Linux system is mounted under /boot/efi.

/home_Partition

The /home Partition

Each user in the system has a home directory to store personal files and preferences, and most of them are located under /home. Usually the home directory is the same as the username, so the user John would have his directory under /home/john.

However there are exceptions. For example the home directory for the root user is /root and some system services may have associated users with home directories elsewhere.

There is no rule to determine the size of a partition for the /home directory (the home partition). You should take into account the number of users in the system and how it will be used. A user which only does web browsing and word processing will require less space than one who works with video editing, for example.

Variable_Data(/var)

This directory contains "variable data", or files and directories the system must be able to write to during operation. This includes system logs (in `/var/log`), temporary files (`/var/tmp`) and cached application data (in `/var/cache`).

`/var/www/html` is also the default directory for the data files for the Apache Web Server and `/var/lib/mysql` is the default location for database files for the MySQL server. However, both of these can be changed.

One good reason for putting `/var` in a separate partition is stability. Many applications and processes write to `/var` and subdirectories, like `/var/log` or `/var/tmp`. A misbehaved process may write data until there is no free space left on the filesystem.

If `/var` is under `/` this may trigger a kernel panic and filesystem corruption, causing a situation that is difficult to recover from. But if `/var` is kept under a separate partition, the root filesystem will be unaffected.

Like in `/home` there is no universal rule to determine the size of a partition for `/var`, as it will vary with how the system is used. On a home system, it may take only a few gigabytes. But on a database or web server much more space may be needed. In such scenarios, it may be wise to put `/var` on a partition on a different disk than the root partition adding an extra layer of protection against physical disk failure.

Swap_Partition

The swap partition is used to swap memory pages from RAM to disk as needed. This partition needs to be of a specific type, and set-up with a proper utility called mkswap before it can be used.

The swap partition cannot be mounted like the others, meaning that you cannot access it like a normal directory and peek at its contents.

A system can have multiple swap partitions (though this is uncommon) and Linux also supports the use of swap files instead of partitions, which can be useful to quickly increase swap space when needed.

The size of the swap partition is a contentious issue. The old rule from the early days of Linux (“twice the amount of RAM”) may not apply anymore depending on how the system is being used and the amount of physical RAM installed.

Logical_Volume_Management(LVM)

LVM
We have already discussed how disks are organized into one or more partitions, with each partition containing a filesystem which describes how files and associated metadata are stored. One of the downsides of partitioning is that the system administrator has to decide beforehand how the available disk space on a device will be distributed. This can present some challenges later, if a partition requires more space than originally planned. Of course partitions can be resized, but this may not be possible if, for example, there is no free space on the disk.

Logical Volume Management (LVM) is a form of storage virtualization that offers system administrators a more flexible approach to managing disk space than traditional partitioning. The goal of LVM is to facilitate managing the storage needs of your end users. The basic unit is the Physical Volume (PV), which is a block device on your system like a disk partition or a RAID array.

PVs are grouped into Volume Groups (VG) which abstract the underlying devices and are seen as a single logical device, with the combined storage capacity of the component PVs.

Each volume in a Volume Group is subdivided into fixed-sized pieces called extents. Extents on a PV are called Physical Extents (PE), while those on a Logical Volume are Logical Extents (LE). Generally, each Logical Extent is mapped to a Physical Extent, but this can change if features like disk mirroring are used.

Volume Groups can be subdivided into Logical Volumes (LVs), which functionally work in a similar way to partitions but with more flexibility.

The size of a Logical Volume, as specified during its creation, is in fact defined by the size of the physical extents (4 MB by default) multiplied by the number of extents on the volume. From this it is easy to understand that to grow a Logical Volume, for example, all that the system administrator has to do is add more extents from the pool available in the Volume Group. Likewise, extents can be removed to shrink the LV.

After a Logical Volume is created it is seen by the operating system as a normal block device. A device will be created in /dev, named as /dev/VGNAME/LVNAME, where VGNAME is the name of the Volume Group, and LVNAME is the name of the Logical Volume.

These devices can be formatted with a desired filesystem using standard utilities (like mkfs.ext4, for example) and mounted using the usual methods, either manually with the mount command or automatically by adding them to the /etc/fstab file.

Install_a_boot_manager

Introduction
When a computer is powered on the first software to run is the boot loader. This is a piece of code whose sole purpose is to load an operating system kernel and hand over control to it. The kernel will load the necessary drivers, initialize the hardware and then load the rest of the operating system.

GRUB is the boot loader used on most Linux distributions. It can load the Linux kernel or other operating systems, such as Windows, and can handle multiple kernel images and parameters as separate menu entries. Kernel selection at boot is done via a keyboard-driven interface, and there is a command-line interface for editing boot options and parameters.

Most Linux distributions install and configure GRUB (actually, GRUB 2) automatically, so a regular user does not

need to think about that. However, as a system administrator, it is vital to know how to control the boot process so you can recover the system from a boot failure after a failed kernel upgrade, for example.

In this lesson you will learn about how to install, configure and interact with GRUB.

GRUB_Legacy_vs_GRUB_2

The original version of GRUB (Grand Unified Bootloader), now known as GRUB Legacy was developed in 1995 as part of the GNU Hurd project, and later was adopted as the default boot loader of many Linux distributions, replacing earlier alternatives such as LILO.

GRUB 2 is a complete rewrite of GRUB aiming to be cleaner, safer, more robust, and more powerful. Among the many advantages over GRUB Legacy are a much more flexible configuration file (with many more commands and conditional statements, similar to a scripting language), a more modular design and better localization/ internationalization.

There is also support for themes and graphical boot menus with splash screens, the ability to boot LiveCD ISOs directly from the hard drive, better support for non-x86 architectures, universal support for UUIDs (making it easier to identify disks and partitions) and much more.

GRUB Legacy is no longer under active development (the last release was 0.97, in 2005), and today most major Linux distributions install GRUB 2 as the default boot loader. However, you may still find systems using GRUB Legacy, so it is important to know how to use it and where it is different from GRUB 2.

Where is the Bootloader?
Historically, hard disks on IBM PC compatible systems were partitioned using the MBR partitioning scheme, created in 1982 for IBM PC-DOS (MS-DOS) 2.0.

In this scheme, the first 512-byte sector of the disk is called the Master Boot Record and contains a table describing the partitions on the disk (the partition table) and also bootstrap code, called a bootloader.

When the computer is turned on, this very minimal (due to size restrictions) bootloader code is loaded, executed and passes control to a secondary boot loader on disk, usually located in a 32 KB space between the MBR and the first partition, which in turn will load the operating system(s).

On an MBR-partitioned disk, the boot code for GRUB is installed to the MBR. This loads and passes control to a “core” image installed between the MBR and the first partition. From this point, GRUB is capable of loading the rest of the needed resources (menu definitions, configuration files and extra modules) from disk.

However, MBR has limitations on the number of partitions (originally a maximum of 4 primary partitions, later a maximum of 3 primary partitions with 1 extended partition subdivided into a number of logical partitions) and maximum disk sizes of 2 TB. To overcome these limitations a new partitioning scheme called GPT (GUID Partition Table), part of the UEFI (Unified Extensible Firmware Interface) standard, was created.

GPT-partitioned disks can be used either with computers with the traditional PC BIOS or ones with UEFI firmware. On machines with a BIOS, the second part of GRUB is stored in a special BIOS boot partition.

On systems with UEFI firmware, GRUB is loaded by the firmware from the files grubia32.efi (for 32-Bit systems) or grubx64.efi (for 64-Bit systems) from a partition called the ESP (EFI System Partition).

The /boot Partition
On Linux the files necessary for the boot process are usually stored on a boot partition, mounted under the root file system and colloquially referred to as /boot.

A boot partition is not needed on current systems, as boot loaders such as GRUB can usually mount the root file system and look for the needed files inside a /boot directory, but it is good practice as it separates the files needed for the boot process from the rest of the filesystem.

This partition is usually the first one on the disk. This is because the original IBM PC BIOS addressed disks using Cylinders, Heads and Sectors (CHS), with a maximum of 1024 cylinders, 256 heads and 63 sectors, resulting in a maximum disk size of 528 MB (504 MB under MS-DOS). This means that anything past this mark would not be accessible, unless a different disk addressing scheme (like LBA, Logical Block Addressing) was used.

So for maximum compatibility, the /boot partition is usually located at the start of the disk and ends before cylinder

1024 (528 MB), ensuring that the machine will always be able to load the kernel. The recommended size for this partition on a current machine is 300 MB.

Other reasons for a separate /boot partition are encryption and compression since some methods may not be supported by GRUB 2 yet, or if you need to have the system root partition (/) formatted using an unsupported file system.

Contents of the Boot Partition
The contents of the /boot partition may vary with system architecture or the boot loader in use, but on a x86-based system you will usually find the files below. Most of these are named with a -VERSION suffix, where -VERSION is the version of the corresponding Linux kernel. So, for example, a configuration file for the Linux kernel version 4.15.0-65-generic would be called config-4.15.0-65-generic.

Config file
This file, usually called config-VERSION (see example above), stores configuration parameters for the Linux kernel. This file is generated automatically when a new kernel is compiled or installed and should not be directly modified by the user.

System map
This file is a look-up table matching symbol names (like variables or functions) to their corresponding position in memory. This is useful when debugging a kind of system failure known as a kernel panic, as it allows the user to know which variable or function was being called when the failure occurred. Like the config file, the name is usually System.map-VERSION (e.g. System.map-4.15.0-65-generic).

Linux kernel
This is the operating system kernel proper. The name is usually vmlinux-VERSION (e.g. vmlinux-4.15.0-65-generic). You may also find the name vmlinuz instead of vmlinux, the z at the end meaning that the file has been compressed.

Initial RAM disk
This is usually called initrd.img-VERSION and contains a minimal root file system loaded into a RAM disk, containing utilities and kernel modules needed so the kernel can mount the real root filesystem.

Boot loader related files
On systems with GRUB installed, these are usually located on /boot/grub and include the GRUB configuration file (/boot/grub/grub.cfg for GRUB 2 or /boot/grub/menu.lst in case of GRUB Legacy), modules (in /boot/grub/i386-pc), translation files (in /boot/grub/locale) and fonts (in /boot/grub/fonts).

Manage_shared_libraries

Introduction
In this lesson we will be discussing shared libraries, also known as shared objects: pieces of compiled, reusable code like functions or classes, that are recurrently used by various programs.

To start with, we will explain what shared libraries are, how to identify them and where they are found. Next, we will go into how to configure their storage locations. Finally, we will show how to search for the shared libraries on which a particular program depends.

Concept of Shared Libraries
Similar to their physical counterparts, software libraries are collections of code that are intended to be used by many different programs; just as physical libraries keep books and other resources to be used by many different people.

To build an executable file from a program’s source code, two important steps are necessary. First, the compiler turns the source code into machine code that is stored in so-called object files. Secondly, the linker combines the object files and links them to libraries in order to generate the final executable file. This linking can be done statically or dynamically. Depending on which method we go for, we will be talking about static libraries or, in case of dynamic linking, about shared libraries. Let us explain their differences.

Static libraries
A static library is merged with the program at link time. A copy of the library code is embedded into the program and becomes part of it. Thus, the program has no dependencies on the library at run time because the program already contains the libraries code. Having no dependencies can be seen as an advantage since you do not have to worry about making sure the used libraries are always available. On the downside, statically linked programs are heavier.

Shared (or dynamic) libraries
In the case of shared libraries, the linker simply takes care that the program references libraries correctly. The linker does, however, not copy any library code into the program file. At run time, though, the shared library must be available to satisfy the program's dependencies. This is an economical approach to managing system resources as it helps reduce the size of program files and only one copy of the library is loaded in memory, even when it is used by multiple programs.

Shared Object File Naming Conventions
The name of a shared library, also known as soname, follows a pattern which is made up of three elements:

Library name (normally prefixed by lib)

so (which stands for "shared object")

Version number of the library

Here an example: libpthread.so.0

By contrast, static library names end in .a, e.g. libpthread.a.

Note
Because the files containing shared libraries must be available when the program is executed, most Linux systems contain shared libraries. Since static libraries are only required in a dedicated file when a program is linked, they might not be present on an end user system.

glibc (GNU C library) is a good example of a shared library. On a Debian GNU/Linux 9.9 system, its file is named libc.so.6. Such rather general file names are normally symbolic links that point to the actual file containing a library, whose name contains the exact version number. In case of glibc, this symbolic link looks like this:

```
$ ls -l /lib/x86_64-linux-gnu/libc.so.6
lrwxrwxrwx 1 root root 12 feb  6 22:17 /lib/x86_64-linux-gnu/libc.so.6 -> libc-2.24.so
```
This pattern of referencing shared library files named by a specific version by more general file names is common practice.

Other examples of shared libraries include libreadline (which allows users to edit command lines as they are typed in and includes support for both Emacs and vi editing modes), libcrypt (which contains functions related to encryption, hashing, and encoding), or libcurl (which is a multiprotocol file transfer library).

Common locations for shared libraries in a Linux system are:

/lib

/lib32

/lib64

/usr/lib

/usr/local/lib

Note
The concept of shared libraries is not exclusive to Linux. In Windows, for example, they are called DLL which stands for dynamic linked libraries.

Configuration of Shared Library Paths
The references contained in dynamically linked programs are resolved by the dynamic linker (ld.so or ld-linux.so) when the program is run. The dynamic linker searches for libraries in a number of directories. These directories are specified by the library path. The library path is configured in the /etc directory, namely in the file /etc/ld.so.conf and, more common nowadays, in files residing in the /etc/ld.so.conf.d directory. Normally, the former includes just a single include line for *.conf files in the latter:

```
$ cat /etc/ld.so.conf
include /etc/ld.so.conf.d/*.conf
```
The /etc/ld.so.conf.d directory contains *.conf files:

```
$ ls /etc/ld.so.conf.d/
libc.conf  x86_64-linux-gnu.conf
```
These *.conf files must include the absolute paths to the shared library directories:

```
$ cat /etc/ld.so.conf.d/x86_64-linux-gnu.conf
# Multiarch support
/lib/x86_64-linux-gnu
/usr/lib/x86_64-linux-gnu
```
The ldconfig command takes care of reading these config files, creating the aforementioned set of symbolic links that help to locate the individual libraries and finally of updating the cache file /etc/ld.so.cache. Thus, ldconfig must be run every time configuration files are added or updated.

Useful options for ldconfig are:

-v, --verbose
Display the library version numbers, the name of each directory, and the links that are created:

```
$ sudo ldconfig -v
/usr/local/lib:
/lib/x86_64-linux-gnu:
    libnss_myhostname.so.2 -> libnss_myhostname.so.2
        libfuse.so.2 -> libfuse.so.2.9.7
        libidn.so.11 -> libidn.so.11.6.16
        libnss_mdns4.so.2 -> libnss_mdns4.so.2
        libparted.so.2 -> libparted.so.2.0.1
        (...)
```
So we can see, for example, how libfuse.so.2 is linked to the actual shared object file libfuse.so.2.9.7.

-p, --print-cache
Print the lists of directories and candidate libraries stored in the current cache:

```
$ sudo ldconfig -p
1094 libs found in the cache `/etc/ld.so.cache'
    libzvbi.so.0 (libc6,x86-64) => /usr/lib/x86_64-linux-gnu/libzvbi.so.0
        libzvbi-chains.so.0 (libc6,x86-64) => /usr/lib/x86_64-linux-gnu/libzvbi-chains.so.0
        libzmq.so.5 (libc6,x86-64) => /usr/lib/x86_64-linux-gnu/libzmq.so.5
        libzeitgeist-2.0.so.0 (libc6,x86-64) => /usr/lib/x86_64-linux-gnu/libzeitgeist-2.0.so.0
        (...)
```
Note how the cache uses the fully qualified soname of the links:

```
$ sudo ldconfig -p |grep libfuse
        libfuse.so.2 (libc6,x86-64) => /lib/x86_64-linux-gnu/libfuse.so.2
```
If we long list /lib/x86_64-linux-gnu/libfuse.so.2, we will find the reference to the actual shared object file libfuse.so.2.9.7 which is stored in the same directory:

```
$ ls -l /lib/x86_64-linux-gnu/libfuse.so.2
lrwxrwxrwx 1 root root 16 Aug 21  2018 /lib/x86_64-linux-gnu/libfuse.so.2 -> libfuse.so.2.9.7
```
Note
Since it requires write access to /etc/ld.so.cache (owned by root), you must either be root or use sudo to invoke ldconfig. For more information about ldconfig switches, refer to its manual page.

In addition to the configuration files described above, the LD_LIBRARY_PATH environment variable can be used to add new paths for shared libraries temporarily. It is made up of a colon-separated (:) set of directories where libraries are looked up. To add, for example, /usr/local/mylib to the library path in the current shell session, you could type:

```
$ LD_LIBRARY_PATH=/usr/local/mylib
```
Now you can check its value:

```
$ echo $LD_LIBRARY_PATH
/usr/local/mylib
```
To add /usr/local/mylib to the library path in the current shell session and have it exported to all child processes spawned from that shell, you would type:

```
$ export LD_LIBRARY_PATH=/usr/local/mylib
```
To remove the LD_LIBRARY_PATH environment variable, just type:

$ unset LD_LIBRARY_PATH
To make the changes permanent, you can write the line

export LD_LIBRARY_PATH=/usr/local/mylib
in one of Bash's initialization scripts such as /etc/bash.bashrc or ~/.bashrc.

Note
LD_LIBRARY_PATH is to shared libraries what PATH is to executables. For more information about environment variables and shell configuration, refer to the respective lessons.

Searching for the Dependencies of a Particular Executable
To look up the shared libraries required by a specific program, use the ldd command followed by the absolute path to the program. The output shows the path of the shared library file as well as the hexadecimal memory address at which it is loaded:

```
$ ldd /usr/bin/git
   linux-vdso.so.1 =>  (0x00007ffcbb310000)
      libpcre.so.3 => /lib/x86_64-linux-gnu/libpcre.so.3 (0x00007f18241eb000)
      libz.so.1 => /lib/x86_64-linux-gnu/libz.so.1 (0x00007f1823fd1000)
      libresolv.so.2 => /lib/x86_64-linux-gnu/libresolv.so.2 (0x00007f1823db6000)
      libpthread.so.0 => /lib/x86_64-linux-gnu/libpthread.so.0 (0x00007f1823b99000)
      librt.so.1 => /lib/x86_64-linux-gnu/librt.so.1 (0x00007f1823991000)
      libc.so.6 => /lib/x86_64-linux-gnu/libc.so.6 (0x00007f18235c7000)
      /lib64/ld-linux-x86-64.so.2 (0x00007f182445b000)
```

Likewise, we use ldd to search for the dependencies of a shared object:

```
$ ldd /lib/x86_64-linux-gnu/libc.so.6
      /lib64/ld-linux-x86-64.so.2 (0x00007fbfed578000)
      linux-vdso.so.1 (0x00007fffb7bf5000)
```

With the -u (or --unused) option ldd prints the unused direct dependencies (if they exist):

```
$ ldd -u /usr/bin/git
Unused direct dependencies:
      /lib/x86_64-linux-gnu/libz.so.1
      /lib/x86_64-linux-gnu/libpthread.so.0
      /lib/x86_64-linux-gnu/librt.so.1
```

The reason for unused dependencies is related to the options used by the linker when building the binary. Although the program does not need an unused library, it was still linked and labelled as NEEDED in the information about the object file. You can investigate this using commands such as readelf or objdump, which you will soon use in the explorational exercise.

Use_Debian_package_management

Introduction
A long time ago, when Linux was still in its infancy, the most common way to distribute software was a compressed file (usually a .tar.gz archive) with source code, which you would unpack and compile yourself.

However, as the amount and complexity of software grew, the need for a way to distribute pre-compiled software became clear. After all, not everyone had the resources, both in time and computing power, to compile large projects like the Linux kernel or an X Server.

Soon, efforts to standardize a way to distribute these software "packages" grew, and the first package managers were born. These tools made it much easier to install, configure or remove software from a system.

One of those was the Debian package format (.deb) and its package tool (dpkg). Today, they are widely used not only on Debian itself, but also on its derivatives, like Ubuntu and those derived from it.

Another package management tool that is popular on Debian-based systems is the Advanced Package Tool (apt), which can streamline many of the aspects of the installation, maintenance and removal of packages, making it much easier.

In this lesson, we will learn how to use both dpkg and apt to obtain, install, maintain and remove software on a

Debian-based Linux system.

The_Debian_Package_Tool(dpkg)

The Debian Package (dpkg) tool is the essential utility to install, configure, maintain and remove software packages on Debian-based systems. The most basic operation is to install a .deb package, which can be done with:

dkpg -i PACKAGENAME
Where PACKAGENAME is the name of the .deb file you want to install.

Package upgrades are handled the same way. Before installing a package, dpkg will check if a previous version already exists in the system. If so, the package will be upgraded to the new version. If not, a fresh copy will be installed.

Dealing with Dependencies
More often than not, a package may depend on others to work as intended. For example, an image editor may need libraries to open JPEG files, or another utility may need a widget toolkit like Qt or GTK for its user interface.

dpkg will check if those dependencies are installed on your system, and will fail to install the package if they are not. In this case, dpkg will list which packages are missing. However it cannot solve dependencies by itself. It is up to the user to find the .deb packages with the corresponding dependencies and install them.

In the example below, the user tries to install the OpenShot video editor package, but some dependencies were missing:

```
# dpkg -i openshot-qt_2.4.3+dfsg1-1_all.deb
(Reading database ... 269630 files and directories currently installed.)
Preparing to unpack openshot-qt_2.4.3+dfsg1-1_all.deb ...
Unpacking openshot-qt (2.4.3+dfsg1-1) over (2.4.3+dfsg1-1) ...
dpkg: dependency problems prevent configuration of openshot-qt:
 openshot-qt depends on fonts-cantarell; however:
  Package fonts-cantarell is not installed.
 openshot-qt depends on python3-openshot; however:
  Package python3-openshot is not installed.
 openshot-qt depends on python3-pyqt5; however:
  Package python3-pyqt5 is not installed.
 openshot-qt depends on python3-pyqt5.qtsvg; however:
  Package python3-pyqt5.qtsvg is not installed.
 openshot-qt depends on python3-pyqt5.qtwebkit; however:
  Package python3-pyqt5.qtwebkit is not installed.
 openshot-qt depends on python3-zmq; however:
  Package python3-zmq is not installed.

dpkg: error processing package openshot-qt (--install):
 dependency problems - leaving unconfigured
Processing triggers for mime-support (3.60ubuntu1) ...
Processing triggers for gnome-menus (3.32.0-1ubuntu1) ...
Processing triggers for desktop-file-utils (0.23-4ubuntu1) ...
Processing triggers for hicolor-icon-theme (0.17-2) ...
Processing triggers for man-db (2.8.5-2) ...
Errors were encountered while processing:
 openshot-qt
```

As shown above, OpenShot depends on the fonts-cantarell, python3-openshot, python3-pyqt5, python3-pyqt5.qtsvg, python3-pyqt5.qtwebkit and python3-zmq packages. All of those need to be installed before the installation of OpenShot can succeed.

Removing_Packages(dpkg)

To remove a package, pass the -r parameter to dpkg, followed by the package name. For example, the following command will remove the unrar package from the system:

```
# dpkg -r unrar
(Reading database ... 269630 files and directories currently installed.)
Removing unrar (1:5.6.6-2) ...
Processing triggers for man-db (2.8.5-2) ...
```

The removal operation also runs a dependency check, and a package cannot be removed unless every other package that depends on it is also removed. If you try to do so, you will get an error message like the one below:

```
# dpkg -r p7zip
dpkg: dependency problems prevent removal of p7zip:
 winetricks depends on p7zip; however:
  Package p7zip is to be removed.
 p7zip-full depends on p7zip (= 16.02+dfsg-6).

dpkg: error processing package p7zip (--remove):
 dependency problems - not removing
Errors were encountered while processing:
 p7zip
```

You can pass multiple package names to dpkg -r, so they will all be removed at once.

When a package is removed, the corresponding configuration files are left on the system. If you want to remove everything associated with the package, use the -P (purge) option instead of -r.

Note
You can force dpkg to install or remove a package, even if dependencies are not met, by adding the --force parameter like in dpkg -i --force PACKAGENAME. However, doing so will most likely leave the installed package, or even your system, in a broken state. Do not use --force unless you are absolutely sure of what you are doing.

Getting_Package_Information(dpkg)

To get information about a .deb package, such as its version, architecture, maintainer, dependencies and more, use the dpkg command with the -I parameter, followed by the filename of the package you want to inspect:

```
# dpkg -I google-chrome-stable_current_amd64.deb
 new Debian package, version 2.0.
 size 59477810 bytes: control archive=10394 bytes.
    1222 bytes,    13 lines      control
   16906 bytes,   457 lines   *  postinst           #!/bin/sh
   12983 bytes,   344 lines   *  postrm             #!/bin/sh
    1385 bytes,    42 lines   *  prerm             #!/bin/sh
 Package: google-chrome-stable
 Version: 76.0.3809.100-1
 Architecture: amd64
 Maintainer: Chrome Linux Team <chromium-dev@chromium.org>
 Installed-Size: 205436
 Pre-Depends: dpkg (>= 1.14.0)
 Depends: ca-certificates, fonts-liberation, libappindicator3-1, libasound2 (>= 1.0.16), libatk-bridge2.0-0 (>=
2.5.3), libatk1.0-0 (>= 2.2.0), libatspi2.0-0 (>= 2.9.90), libc6 (>= 2.16), libcairo2 (>= 1.6.0), libcups2 (>= 1.4.0),
libdbus-1-3 (>= 1.5.12), libexpat1 (>= 2.0.1), libgcc1 (>= 1:3.0), libgdk-pixbuf2.0-0 (>= 2.22.0), libglib2.0-0 (>=
2.31.8), libgtk-3-0 (>= 3.9.10), libnspr4 (>= 2:4.9-2~), libnss3 (>= 2:3.22), libpango-1.0-0 (>= 1.14.0),
libpangocairo-1.0-0 (>= 1.14.0), libuuid1 (>= 2.16), libx11-6 (>= 2:1.4.99.1), libx11-xcb1, libxcb1 (>= 1.6),
libxcomposite1 (>= 1:0.3-1), libxcursor1 (>> 1.1.2), libxdamage1 (>= 1:1.1), libxext6, libxfixes3, libxi6 (>=
2:1.2.99.4), libxrandr2 (>= 2:1.2.99.3), libxrender1, libxss1, libxtst6, lsb-release, wget, xdg-utils (>= 1.0.2)
 Recommends: libu2f-udev
 Provides: www-browser
 Section: web
 Priority: optional
 Description: The web browser from Google
  Google Chrome is a browser that combines a minimal design with sophisticated technology to make the web
faster, safer, and easier.
```

Listing_Installed_Packages_and_Package_Contents(dpkg)

To get a list of every package installed on your system, use the --get-selections option, as in dpkg --get-selections. You can also get a list of every file installed by a specific package by passing the -L PACKAGENAME parameter to dpkg, like below:

```
# dpkg -L unrar
/.
/usr
/usr/bin
/usr/bin/unrar-nonfree
/usr/share
/usr/share/doc
/usr/share/doc/unrar
/usr/share/doc/unrar/changelog.Debian.gz
/usr/share/doc/unrar/copyright
/usr/share/man
/usr/share/man/man1
/usr/share/man/man1/unrar-nonfree.1.gz
```

Reconfiguring_Installed_Package(dpkg)

When a package is installed there is a configuration step called post-install where a script runs to set-up everything needed for the software to run such as permissions, placement of configuration files, etc. This may also ask some questions of the user to set preferences on how the software will run.

Sometimes, due to a corrupt or malformed configuration file, you may wish to restore a package's settings to its "fresh" state. Or you may wish to change the answers you gave to the initial configuration questions. To do this run the dpkg-reconfigure utility, followed by the package name.

This program will back-up the old configuration files, unpack the new ones in the correct directories and run the post-install script provided by the package, as if the package had been installed for the first time. Try to reconfigure the tzdata package with the following example:

```
# dpkg-reconfigure tzdata
```

Advanced_Package_Tool(apt)

The Advanced Package Tool (APT) is a package management system, including a set of tools, that greatly simplifies package installation, upgrade, removal and management. APT provides features like advanced search capabilities and automatic dependency resolution.

APT is not a "substitute" for dpkg. You may think of it as a "front end", streamlining operations and filling gaps in dpkg functionality, like dependency resolution.

APT works in concert with software repositories which contain the packages available to install. Such repositories may be a local or remote server, or (less common) even a CD-ROM disc.

Linux distributions, such as Debian and Ubuntu, maintain their own repositories, and other repositories may be maintained by developers or user groups to provide software not available from the main distribution repositories.

There are many utilities that interact with APT, the main ones being:

apt-get
used to download, install, upgrade or remove packages from the system.

apt-cache
used to perform operations, like searches, in the package index.

apt-file
used for searching for files inside packages.

There is also a “friendlier” utility named simply apt, combining the most used options of apt-get and apt-cache in one utility. Many of the commands for apt are the same as the ones for apt-get, so they are in many cases interchangeable. However, since apt may not be installed on a system, it is recommended to learn how to use apt-get and apt-cache.

Note
apt and apt-get may require a network connection, because packages and package indexes may need to be downloaded from a remote server.

Updating_the_Package_Index(apt)

Before installing or upgrading software with APT, it is recommended to update the package index first in order to retrieve information about new and updated packages. This is done with the apt-get command, followed by the update parameter:

```
# apt-get update
Ign:1 http://dl.google.com/linux/chrome/deb stable InRelease
Hit:2 https://repo.skype.com/deb stable InRelease
Hit:3 http://us.archive.ubuntu.com/ubuntu disco InRelease
Hit:4 http://repository.spotify.com stable InRelease
Hit:5 http://dl.google.com/linux/chrome/deb stable Release
Hit:6 http://apt.pop-os.org/proprietary disco InRelease
Hit:7 http://ppa.launchpad.net/system76/pop/ubuntu disco InRelease
Hit:8 http://us.archive.ubuntu.com/ubuntu disco-security InRelease
Hit:9 http://us.archive.ubuntu.com/ubuntu disco-updates InRelease
Hit:10 http://us.archive.ubuntu.com/ubuntu disco-backports InRelease
Reading package lists... Done
```

Tip
Instead of apt-get update, you can also use apt update.

Installing_and_Removing_Packages(apt)

With the package index updated you may now install a package. This is done with apt-get install, followed by the name of the package you wish to install:

```
# apt-get install xournal
Reading package lists... Done
Building dependency tree
Reading state information... Done
The following NEW packages will be installed:
  xournal
0 upgraded, 1 newly installed, 0 to remove and 75 not upgraded.
Need to get 285 kB of archives.
After this operation, 1041 kB of additional disk space will be used.
```

Similarly, to remove a package use apt-get remove, followed by the package name:

```
# apt-get remove xournal
Reading package lists... Done
Building dependency tree
Reading state information... Done
The following packages will be REMOVED:
  xournal
0 upgraded, 0 newly installed, 1 to remove and 75 not upgraded.
After this operation, 1041 kB disk space will be freed.
Do you want to continue? [Y/n]
```

Be aware that when installing or removing packages, APT will do automatic dependency resolution. This means that any additional packages needed by the package you are installing will also be installed, and that packages that depend on the package you are removing will also be removed. APT will always show what will be installed or removed before asking if you want to continue:

```
# apt-get remove p7zip
Reading package lists... Done
Building dependency tree
The following packages will be REMOVED:
  android-libbacktrace android-libunwind android-libutils
  android-libziparchive android-sdk-platform-tools fastboot p7zip p7zip-full
0 upgraded, 0 newly installed, 8 to remove and 75 not upgraded.
After this operation, 6545 kB disk space will be freed.
Do you want to continue? [Y/n]
```

Note that when a package is removed the corresponding configuration files are left on the system. To remove the package and any configuration files, use the purge parameter instead of remove or the remove parameter with the --purge option:

```
# apt-get purge p7zip
```

or

```
# apt-get remove --purge p7zip
```

Tip
You can also use apt install and apt remove.

Fixing_Broken_Dependencies(apt)

It is possible to have “broken dependencies” on a system. This means that one or more of the installed packages depend on other packages that have not been installed, or are not present anymore. This may happen due to an APT error, or because of a manually installed package.

To solve this, use the apt-get install -f command. This will try to “fix” the broken packages by installing the missing dependencies, ensuring that all packages are consistent again.

Tip
You can also use apt install -f.

Upgrading_Packages(apt)

APT can be used to automatically upgrade any installed packages to the latest versions available from the repositories. This is done with the apt-get upgrade command. Before running it, first update the package index with apt-get update:

```
# apt-get update
Hit:1 http://us.archive.ubuntu.com/ubuntu disco InRelease
Hit:2 http://us.archive.ubuntu.com/ubuntu disco-security InRelease
Hit:3 http://us.archive.ubuntu.com/ubuntu disco-updates InRelease
Hit:4 http://us.archive.ubuntu.com/ubuntu disco-backports InRelease
Reading package lists... Done

# apt-get upgrade
Reading package lists... Done
Building dependency tree
Reading state information... Done
Calculating upgrade... Done
The following packages have been kept back:
  gnome-control-center
The following packages will be upgraded:
  cups cups-bsd cups-client cups-common cups-core-drivers cups-daemon
  cups-ipp-utils cups-ppdc cups-server-common firefox-locale-ar (...)

74 upgraded, 0 newly installed, 0 to remove and 1 not upgraded.
Need to get 243 MB of archives.
After this operation, 30.7 kB of additional disk space will be used.
Do you want to continue? [Y/n]
```

The summary at the bottom of the output shows how many packages will be upgraded, how many will be installed,

removed or kept back, the total download size and how much extra disk space will be needed to complete the operation. To complete the upgrade, just answer Y and wait for apt-get to finish the task.

To upgrade a single package, just run apt-get upgrade followed by the package name. As in dpkg, apt-get will first check if a previous version of a package is installed. If so, the package will be upgraded to the newest version available in the repository. If not, a fresh copy will be installed.

Tip
You can also use apt upgrade and apt update.

The_Local_Cache(apt)

When you install or update a package, the corresponding .deb file is downloaded to a local cache directory before the package is installed. By default, this directory is /var/cache/apt/archives. Partially downloaded files are copied to /var/cache/apt/archives/partial/.

As you install and upgrade packages, the cache directory can get quite large. To reclaim space, you can empty the cache by using the apt-get clean command. This will remove the contents of the /var/cache/apt/archives and /var/cache/apt/archives/partial/ directories.

Tip
You can also use apt clean.

Searching for Packages(apt)

The apt-cache utility can be used to perform operations on the package index, such as searching for a specific package or listing which packages contain a specific file.

To conduct a search, use apt-cache search followed by a search pattern. The output will be a list of every package that contains the pattern, either in its package name, description or files provided.

```
# apt-cache search p7zip
liblzma-dev - XZ-format compression library - development files
liblzma5 - XZ-format compression library
forensics-extra - Forensics Environment - extra console components (metapackage)
p7zip - 7zr file archiver with high compression ratio
p7zip-full - 7z and 7za file archivers with high compression ratio
p7zip-rar - non-free rar module for p7zip
```

In the example above, the entry liblzma5 - XZ-format compression library does not seem to match the pattern. However, if we show the full information, including description, for the package using the show parameter, we will find the pattern there:

```
# apt-cache show liblzma5
Package: liblzma5
Architecture: amd64
Version: 5.2.4-1
Multi-Arch: same
Priority: required
Section: libs
Source: xz-utils
Origin: Ubuntu
Maintainer: Ubuntu Developers <ubuntu-devel-discuss@lists.ubuntu.com>
Original-Maintainer: Jonathan Nieder <jrnieder@gmail.com>
Bugs: https://bugs.launchpad.net/ubuntu/+filebug
Installed-Size: 259
Depends: libc6 (>= 2.17)
Breaks: liblzma2 (<< 5.1.1alpha+20110809-3~)
Filename: pool/main/x/xz-utils/liblzma5_5.2.4-1_amd64.deb
Size: 92352
MD5sum: 223533a347dc76a8cc9445cfc6146ec3
SHA1: 8ed14092fb1caecfebc556fda0745e1e74ba5a67
SHA256: 01020b5a0515dbc9a7c00b464a65450f788b0258c3fbb733ecad0438f5124800
Homepage: https://tukaani.org/xz/
```

Description-en: XZ-format compression library
 XZ is the successor to the Lempel-Ziv/Markov-chain Algorithm
 compression format, which provides memory-hungry but powerful
 compression (often better than bzip2) and fast, easy decompression.
 .
 The native format of liblzma is XZ; it also supports raw (headerless)
 streams and the older LZMA format used by lzma. (For 7-Zip's related
 format, use the p7zip package instead.)
You can use regular expressions with the search pattern, allowing for very complex (and precise) searches. However, this topic is out of scope for this lesson.

Tip
You can also use apt search instead of apt-cache search and apt show instead of apt-cache show.

The Sources List

APT uses a list of sources to know where to get packages from. This list is stored in the file sources.list, located inside the /etc/apt directory. This file can be edited directly with a text editor, like vi, pico or nano, or with graphical utilities like aptitude or synaptic.

A typical line inside sources.list looks like this:

deb http://us.archive.ubuntu.com/ubuntu/ disco main restricted universe multiverse
The syntax is archive type, URL, distribution and one or more components, where:

Archive type
A repository may contain packages with ready-to-run software (binary packages, type deb) or with the source code to this software (source packages, type deb-src). The example above provides binary packages.

URL
The URL for the repository.

Distribution
The name (or codename) for the distribution for which packages are provided. One repository may host packages for multiple distributions. In the example above, disco is the codename for Ubuntu 19.04 Disco Dingo.

Components
Each component represents a set of packages. These components may be different on different Linux distributions. For example, on Ubuntu and derivatives, they are:

main
contains officially supported, open-source packages.

restricted
contains officially supported, closed-source software, like device drivers for graphic cards, for example.

universe
contains community maintained open-source software.

multiverse
contains unsupported, closed-source or patent-encumbered software.

On Debian, the main components are:

main
consists of packages compliant with the Debian Free Software Guidelines (DFSG), which do not rely on software outside this area to operate. Packages included here are considered to be part of the Debian distribution.

contrib
contains DFSG-compliant packages, but which depend on other packages that are not in main.

non-free
contains packages that are not compliant with the DFSG.

security
contains security updates.

backports
contains more recent versions of packages that are in main. The development cycle of the stable versions of Debian is quite long (around two years), and this ensures that users can get the most up-to-date packages without having to modify the main core repository.

Note
You can learn more about the Debian Free Software Guidelines at: https://www.debian.org/social_contract#guidelines

To add a new repository to get packages from, you can simply add the corresponding line (usually provided by the repository maintainer) to the end of sources.list, save the file and reload the package index with apt-get update. After that, the packages in the new repository will be available for installation using apt-get install.

Keep in mind that lines starting with the # character are considered comments, and are ignored.

The /etc/apt/sources.list.d Directory
Inside the /etc/apt/sources.list.d directory you can add files with additional repositories to be used by APT, without the need to modify the main /etc/apt/sources.list file. These are simple text files, with the same syntax described above and the .list file extension.

Below you see the contents of a file called /etc/apt/sources.list.d/buster-backports.list:

deb http://deb.debian.org/debian buster-backports main contrib non-free
deb-src http://deb.debian.org/debian buster-backports main contrib non-free
Listing Package Contents and Finding Files
A utility called apt-file can be used to perform more operations in the package index, like listing the contents of a package or finding a package that contains a specific file. This utility may not be installed by default in your system. In this case, you can usually install it using apt-get:

apt-get install apt-file
After installation, you will need to update the package cache used for apt-file:

apt-file update
This usually takes only a few seconds. After that, you are ready to use apt-file.

To list the contents of a package, use the list parameter followed by the package name:

apt-file list unrar
unrar: /usr/bin/unrar-nonfree
unrar: /usr/share/doc/unrar/changelog.Debian.gz
unrar: /usr/share/doc/unrar/copyright
unrar: /usr/share/man/man1/unrar-nonfree.1.gz
Tip
You can also use apt list instead of apt-file list.

You can search all packages for a file using the search parameter, followed by the file name. For example, if you wish to know which package provides a file called libSDL2.so, you can use:

apt-file search libSDL2.so
libsdl2-dev: /usr/lib/x86_64-linux-gnu/libSDL2.so
The answer is the package libsdl2-dev, which provides the file /usr/lib/x86_64-linux-gnu/libSDL2.so.

The difference between apt-file search and dpkg-query is that apt-file search will also search uninstalled packages, while dpkg-query can only show files that belong to an installed package.