

LPIC-1 - 101-500

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)
```

```
2/3104:0b.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

3/31bDeviceClass

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/lp, 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.
4/31Size
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-filefilesystems 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/fd0`, `/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.

7/31It 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.

8/31Runlevel 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
9/31# 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
```

```
N3
```

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

10/31unit 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 start 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 `runlevel0.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.

11/31Another 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`:

```
12/31# 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
```

```
[PCSP,TRISTATE,COMPAT,EPP,ECP,DMA]
```

```
[ 5.800146] lp0: using parport0 (interrupt-driven).
```

```
[ 5.897421] systemd-journal[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
```

```
13/31-1 55c0d9439bfb4e85a20a62776d0dbb4d Thu 2019-10-03 19:27:53 -03—Fri 2019-10-04 00:28:47 -03
```

```
0 08fbbbebd9f964a74b8a02bb27b200622 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-journal[352]: Journal started
```

```
oct 04 00:31:01 ubuntu-host systemd-journal[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

14/31Ctrl+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 you 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. 17/31The 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 19/311024 (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.

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)install

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:

```
# dpkg -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:

```
24/31# 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 -l parameter, followed by the filename of the package you want to inspect:

```
# dpkg -l 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, libxfixed3, 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(dp

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
```

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.

26/31There 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:

```
27/31# 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,

28/31 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/
```

```
29/31Description-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.

`30/31security`

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.

LPIC-1 - 101-500 - Continue

Use RPM and YUM 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 as 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 RPM Package Manager and its corresponding tool (rpm), developed by Red Hat. Today, they are widely used not only on Red Hat Enterprise Linux (RHEL) itself, but also on its descendants, like Fedora, CentOS and Oracle Linux, other distributions like openSUSE and even other operating systems, like IBM’S AIX.

Other package management tools popular on Red Hat compatible distros are yum (YellowDog Updater Modified), dnf (Dandified YUM) and zypper, which can streamline many of the aspects of the installation, maintenance and removal of packages, making package management much easier.

In this lesson, we will learn how to use rpm, yum, dnf and zypper to obtain, install, manage and remove software on a Linux system.

Note

Despite using the same package format, there are internal differences between distributions so a package made specifically for openSUSE might not work on a RHEL system, and vice-versa. When searching for packages, always check for compatibility and try to find one tailored for your specific distribution, if possible.

The RPM Package Manager (rpm)

The RPM Package Manager (rpm) is the essential tool for managing software packages on Red Hat-based (or derived) systems.

Installing, Upgrading and Removing Packages(rpm)

The most basic operation is to install a package, which can be done with:

```
# rpm -i PACKAGENAME
```

Where PACKAGENAME is the name of the .rpm package you wish to install.

If there is a previous version of a package on the system, you can upgrade to a newer version using the -U parameter:

```
# rpm -U PACKAGENAME
```


If there is no previous version of PACKAGENAME installed, then a fresh copy will be installed. To avoid this and only upgrade an installed package, use the -F option.

In both operations you can add the -v parameter to get a verbose output (more information is shown during the installation) and -h to get hash signs (#) printed as a visual aid to track installation progress. Multiple parameters can be combined into one, so rpm -i -v -h is the same as rpm -ivh.

To remove an installed package, pass the -e parameter (as in “erase”) to rpm, followed by the name of the package you wish to remove:

```
# rpm -e wget
```

If an installed package depends on the package being removed, you will get an error message:

```
# rpm -e unzip
```

error: Failed dependencies:

/usr/bin/unzip is needed by (installed) file-roller-3.28.1-2.el7.x86_64

To complete the operation, first you will need to remove the packages that depend on the one you wish to remove (in the example above, file-roller). You can pass multiple package names to rpm -e to remove multiple packages at once.

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 JPG files, or a utility may need a widget toolkit like Qt or GTK for its user interface.

rpm will check if those dependencies are installed on your system, and will fail to install the package if they are not. In this case, rpm will list what is missing. However it cannot solve dependencies by itself.

In the example below, the user tried to install a package for the GIMP image editor, but some dependencies were missing:

```
# rpm -i gimp-2.8.22-1.el7.x86_64.rpm
```

error: Failed dependencies:

babl(x86-64) >= 0.1.10 is needed by gimp-2:2.8.22-1.el7.x86_64

gegl(x86-64) >= 0.2.0 is needed by gimp-2:2.8.22-1.el7.x86_64

gimp-libs(x86-64) = 2:2.8.22-1.el7 is needed by gimp-2:2.8.22-1.el7.x86_64

libbabl-0.1.so.0()(64bit) is needed by gimp-2:2.8.22-1.el7.x86_64

libgegl-0.2.so.0()(64bit) is needed by gimp-2:2.8.22-1.el7.x86_64

libgimp-2.0.so.0()(64bit) is needed by gimp-2:2.8.22-1.el7.x86_64

libgimpbase-2.0.so.0()(64bit) is needed by gimp-2:2.8.22-1.el7.x86_64

libgimpcolor-2.0.so.0()(64bit) is needed by gimp-2:2.8.22-1.el7.x86_64

libgimpconfig-2.0.so.0()(64bit) is needed by gimp-2:2.8.22-1.el7.x86_64

libgimpmath-2.0.so.0()(64bit) is needed by gimp-2:2.8.22-1.el7.x86_64

libgimpmodule-2.0.so.0()(64bit) is needed by gimp-2:2.8.22-1.el7.x86_64

libgimpthumb-2.0.so.0()(64bit) is needed by gimp-2:2.8.22-1.el7.x86_64

libgimpui-2.0.so.0()(64bit) is needed by gimp-2:2.8.22-1.el7.x86_64

libgimpwidgets-2.0.so.0()(64bit) is needed by gimp-2:2.8.22-1.el7.x86_64

libmng.so.1()(64bit) is needed by gimp-2:2.8.22-1.el7.x86_64

libwmf-0.2.so.7()(64bit) is needed by gimp-2:2.8.22-1.el7.x86_64

libwmflite-0.2.so.7()(64bit) is needed by gimp-2:2.8.22-1.el7.x86_64

It is up to the user to find the .rpm packages with the corresponding dependencies and install them. Package managers such as yum, zypper and dnf have tools that can tell which package provides a specific file. Those will be discussed later in this lesson.

Listing Installed Packages(rpm)

To get a list of all installed packages on your system, use the rpm -qa (think of “query all”).

```
# rpm -qa
selinux-policy-3.13.1-229.el7.noarch
pciutils-libs-3.5.1-3.el7.x86_64
redhat-menus-12.0.2-8.el7.noarch
grubby-8.28-25.el7.x86_64
hunspell-en-0.20121024-6.el7.noarch
dejavu-fonts-common-2.33-6.el7.noarch
xorg-x11-drv-dummy-0.3.7-1.el7.1.x86_64
libevdev-1.5.6-1.el7.x86_64
[...]
```

Getting Package Information(rpm)

To get information about an installed package, such as its version number, architecture, install date, packager, summary, etc., use rpm with the -qi (think of “query info”) parameters, followed by the package name. For example:

```
# rpm -qi unzip
Name       : unzip
Version    : 6.0
Release    : 19.el7
Architecture: x86_64
Install Date: Sun 25 Aug 2019 05:14:39 PM EDT
Group      : Applications/Archiving
Size       : 373986
License    : BSD
Signature  : RSA/SHA256, Wed 25 Apr 2018 07:50:02 AM EDT, Key ID 24c6a8a7f4a80eb5
Source RPM : unzip-6.0-19.el7.src.rpm
Build Date : Wed 11 Apr 2018 01:24:53 AM EDT
Build Host : x86-01.bsys.centos.org
Relocations : (not relocatable)
Packager   : CentOS BuildSystem <http://bugs.centos.org>
Vendor     : CentOS
URL        : http://www.info-zip.org/UnZip.html
Summary    : A utility for unpacking zip files
Description :
The unzip utility is used to list, test, or extract files from a zip
archive. Zip archives are commonly found on MS-DOS systems. The zip
utility, included in the zip package, creates zip archives. Zip and
unzip are both compatible with archives created by PKWARE(R)'s PKZIP
for MS-DOS, but the programs' options and default behaviors do differ
in some respects.
```

Install the unzip package if you need to list, test or extract files from a zip archive.

To get a list of what files are inside an installed package use the -ql parameters (think of “query list”) followed by the package name:

```
# rpm -ql unzip
/usr/bin/funzip
```

```
/usr/bin/unzip
/usr/bin/unzipsfx
/usr/bin/zipgrep
/usr/bin/zipinfo
/usr/share/doc/unzip-6.0
/usr/share/doc/unzip-6.0/BUGS
/usr/share/doc/unzip-6.0/LICENSE
/usr/share/doc/unzip-6.0/README
/usr/share/man/man1/funzip.1.gz
/usr/share/man/man1/unzip.1.gz
/usr/share/man/man1/unzipsfx.1.gz
/usr/share/man/man1/zipgrep.1.gz
/usr/share/man/man1/zipinfo.1.gz
```

If you wish to get information or a file listing from a package that has not been installed yet, just add the -p parameter to the commands above, followed by the name of the RPM file (FILENAME). So rpm -qi PACKAGENAME becomes rpm -qip FILENAME, and rpm -ql PACKAGENAME becomes rpm -qlp FILENAME, as shown below.

```
# rpm -qip atom.x86_64.rpm
Name      : atom
Version   : 1.40.0
Release   : 0.1
Architecture: x86_64
Install Date: (not installed)
Group     : Unspecified
Size      : 570783704
License   : MIT
Signature : (none)
Source RPM : atom-1.40.0-0.1.src.rpm
Build Date : sex 09 ago 2019 12:36:31 -03
Build Host : b01bbeaf3a88
Relocations : /usr
URL        : https://atom.io/
Summary    : A hackable text editor for the 21st Century.
Description :
A hackable text editor for the 21st Century.
# rpm -qlp atom.x86_64.rpm
/usr/bin/apm
/usr/bin/atom
/usr/share/applications/atom.desktop
/usr/share/atom
/usr/share/atom/LICENSE
/usr/share/atom/LICENSES.chromium.html
/usr/share/atom/atom
/usr/share/atom/atom.png
/usr/share/atom/blink_image_resources_200_percent.pak
/usr/share/atom/content_resources_200_percent.pak
/usr/share/atom/content_shell.pak
```

(listing goes on)

Finding Out Which Package Owns a Specific File(rpm)

To find out which installed package owns a file, use the `-qf` (think “query file”) followed by the full path to the file:

```
# rpm -qf /usr/bin/unzip
unzip-6.0-19.el7.x86_64
```

In the example above, the file `/usr/bin/unzip` belongs to the `unzip-6.0-19.el7.x86_64` package.

YellowDog Updater Modified (YUM)

`yum` was originally developed as the Yellow Dog Updater (YUP), a tool for package management on the Yellow Dog Linux distribution. Over time, it evolved to manage packages on other RPM based systems, such as Fedora, CentOS, Red Hat Enterprise Linux and Oracle Linux.

Functionally, it is similar to the `apt` utility on Debian-based systems, being able to search for, install, update and remove packages and automatically handle dependencies. `yum` can be used to install a single package, or to upgrade a whole system at once.

Searching for Packages(yum)

In order to install a package, you need to know its name. For this you can perform a search with `yum search PATTERN`, where `PATTERN` is the name of the package you are searching for. The result is a list of packages whose name or summary contain the search pattern specified. For example, if you need a utility to handle 7Zip compressed files (with the `.7z` extension) you can use:

```
# yum search 7zip
```

```
Loaded plugins: fastestmirror, langpacks
```

```
Loading mirror speeds from cached hostfile
```

```
* base: mirror.ufscar.br
```

```
* epel: mirror.globo.com
```

```
* extras: mirror.ufscar.br
```

```
* updates: mirror.ufscar.br
```

```
===== N/S matchyutr54ed: 7zip
```

```
=====
```

```
p7zip-plugins.x86_64 : Additional plugins for p7zip
```

```
p7zip.x86_64 : Very high compression ratio file archiver
```

```
p7zip-doc.noarch : Manual documentation and contrib directory
```

```
p7zip-gui.x86_64 : 7zG - 7-Zip GUI version
```

Name and summary matches only, use "search all" for everything.

Installing, Upgrading and Removing Packages(yum)

To install a package using `yum`, use the command `yum install PACKAGENAME`, where `PACKAGENAME` is the name of the package. `yum` will fetch the package and corresponding dependencies from an online repository, and install everything in your system.

```
# yum install p7zip
Loaded plugins: fastestmirror, langpacks
Loading mirror speeds from cached hostfile
* base: mirror.ufscar.br
* epel: mirror.globo.com
* extras: mirror.ufscar.br
* updates: mirror.ufscar.br
Resolving Dependencies
--> Running transaction check
---> Package p7zip.x86_64 0:16.02-10.el7 will be installed
--> Finished Dependency Resolution
```

Dependencies Resolved

```
=====
Package      Arch      Version      Repository    Size
=====
Installing:
p7zip        x86_64    16.02-10.el7    epel          604 k
```

Transaction Summary

```
=====
Install 1 Package
```

Total download size: 604 k

Installed size: 1.7 M

Is this ok [y/d/N]:

To upgrade an installed package, use yum update PACKAGENAME, where PACKAGENAME is the name of the package you want to upgrade. For example:

```
# yum update wget
Loaded plugins: fastestmirror, langpacks
Loading mirror speeds from cached hostfile
* base: mirror.ufscar.br
* epel: mirror.globo.com
* extras: mirror.ufscar.br
* updates: mirror.ufscar.br
Resolving Dependencies
--> Running transaction check
---> Package wget.x86_64 0:1.14-18.el7 will be updated
---> Package wget.x86_64 0:1.14-18.el7_6.1 will be an update
--> Finished Dependency Resolution
```

Dependencies Resolved

```
=====
Package      Arch      Version      Repository    Size
=====
Updating:
wget         x86_64    1.14-18.el7_6.1    updates      547 k
```

Transaction Summary

```
=====
Upgrade 1 Package
```

Total download size: 547 k

Is this ok [y/d/N]:

If you omit the name of a package, you can update every package on the system for which an update is available.

To check if an update is available for a specific package, use `yum check-update PACKAGENAME`. As before, if you omit the package name, yum will check for updates for every installed package on the system.

To remove an installed package, use `yum remove PACKAGENAME`, where `PACKAGENAME` is the name of the package you wish to remove.

Finding Which Package Provides a Specific File(yum)

In a previous example we showed an attempt to install the gimp image editor, which failed because of unmet dependencies. However, rpm shows which files are missing, but does not list the name of the packages that provide them.

For example, one of the dependencies missing was `libgimpui-2.0.so.0`. To see what package provides it, you can use `yum whatprovides`, followed by the name of the file you are searching for:

```
# yum whatprovides libgimpui-2.0.so.0
Loaded plugins: fastestmirror, langpacks
Loading mirror speeds from cached hostfile
* base: mirror.ufscar.br
* epel: mirror.globo.com
* extras: mirror.ufscar.br
* updates: mirror.ufscar.br
2:gimp-libs-2.8.22-1.el7.i686 : GIMP libraries
Repo      : base
Matched from:
Provides   : libgimpui-2.0.so.0
The answer is gimp-libs-2.8.22-1.el7.i686. You can then install the package with the command yum
install gimp-libs.
```

This also works for files already in your system. For example, if you wish to know where the file `/etc/hosts` came from, you can use:

```
# yum whatprovides /etc/hosts
Loaded plugins: fastestmirror, langpacks
Loading mirror speeds from cached hostfile
* base: mirror.ufscar.br
* epel: mirror.globo.com
* extras: mirror.ufscar.br
* updates: mirror.ufscar.br
setup-2.8.71-10.el7.noarch : A set of system configuration and setup files
Repo      : base
Matched from:
Filename   : /etc/hosts
The answer is setup-2.8.71-10.el7.noarch.
```

Getting Information About a Package(yum)

To get information about a package, such as its version, architecture, description, size and more, use `yum info PACKAGENAME` where `PACKAGENAME` is the name of the package you want information for:

```
# yum info firefox
Last metadata expiration check: 0:24:16 ago on Sat 21 Sep 2019 02:39:43 PM -03.
Installed Packages
Name      : firefox
Version   : 69.0.1
Release   : 3.fc30
Architecture : x86_64
Size      : 268 M
Source    : firefox-69.0.1-3.fc30.src.rpm
Repository : @System
From repo  : updates
Summary   : Mozilla Firefox Web browser
URL       : https://www.mozilla.org/firefox/
License   : MPLv1.1 or GPLv2+ or LGPLv2+
Description : Mozilla Firefox is an open-source web browser, designed
            : for standards compliance, performance and portability.
Ma
```

Managing Software Repositories(yum)

For yum the “repos” are listed in the directory `/etc/yum.repos.d/`. Each repository is represented by a `.repo` file, like `CentOS-Base.repo`.

Additional, extra repositories can be added by the user by adding a `.repo` file in the directory mentioned above, or at the end of `/etc/yum.conf`. However, the recommended way to add or manage repositories is with the `yum-config-manager` tool.

To add a repository, use the `--add-repo` parameter, followed by the URL to a `.repo` file.

```
# yum-config-manager --add-repo https://rpms.remirepo.net/enterprise/remi.repo
Loaded plugins: fastestmirror, langpacks
adding repo from: https://rpms.remirepo.net/enterprise/remi.repo
grabbing file https://rpms.remirepo.net/enterprise/remi.repo to /etc/yum.repos.d/remi.repo
repo saved to /etc/yum.repos.d/remi.repo
To get a list of all available repositories use yum repolist all. You will get an output similar to this:
```

```
# yum repolist all
Loaded plugins: fastestmirror, langpacks
Loading mirror speeds from cached hostfile
* base: mirror.ufscar.br
* epel: mirror.globo.com
* extras: mirror.ufscar.br
* updates: mirror.ufscar.br
repo id          repo name          status
updates/7/x86_64 CentOS-7 - Updates enabled: 2,500
updates-source/7 CentOS-7 - Updates Sources disabled
disabled repositories will be ignored when installing or upgrading software. To enable or disable a repository, use the yum-config-manager utility, followed by the repository id.
```

In the output above, the repository id is shown on the first column (repo id) of each line. Use only the part before the first /, so the id for the CentOS-7 - Updates repo is updates, and not updates/7/x86_64.

```
# yum-config-manager --disable updates
```

The command above will disable the updates repo. To re-enable it use:

```
# yum-config-manager --enable updates
```

Note

Yum stores downloaded packages and associated metadata in a cache directory (usually /var/cache/yum). As the system gets upgraded and new packages are installed, this cache can get quite large. To clean the cache and reclaim disk space you can use the yum clean command, followed by what to clean. The most useful parameters are packages (yum clean packages) to delete downloaded packages and metadata (yum clean metadata) to delete associated metadata. See the manual page for yum (type man yum) for more information.

DNF

dnf is the package management tool used on Fedora, and is a fork of yum. As such, many of the commands and parameters are similar. This section will give you just a quick overview of dnf.

Searching for packages

dnf search PATTERN, where PATTERN is what you are searching for. For example, dnf search unzip will show all packages that contain the word unzip in the name or description.

Getting information about a package

```
dnf info PACKAGENAME
```

Installing packages

dnf install PACKAGENAME, where PACKAGENAME is the name of the package you wish to install. You can find the name by performing a search.

Removing packages

```
dnf remove PACKAGENAME
```

Upgrading packages

dnf upgrade PACKAGENAME to update only one package. Omit the package name to upgrade all the packages in the system.

Finding out which package provides a specific file

```
dnf provides FILENAME
```

Getting a list of all the packages installed in the system

```
dnf list --installed
```

Listing the contents of a package

```
dnf repoquery -l PACKAGENAME
```

Note

dnf has a built-in help system, which shows more information (such as extra parameters) for each command. To use it, type dnf help followed by the command, like dnf help install.

Managing Software Repositories

Just as with yum and zypper, dnf works with software repositories (repos). Each distribution has a list of default repositories, and administrators can add or remove repos as needed.

To get a list of all available repositories, use `dnf repolist`. To list only enabled repositories, add the `--enabled` option, and to list only disabled repositories, add the `--disabled` option.

```
# dnf repolist
```

Last metadata expiration check: 0:20:09 ago on Sat 21 Sep 2019 02:39:43 PM -03.

repo id	repo name	status
*fedora	Fedora 30 - x86_64	56,582
*fedora-modular	Fedora Modular 30 - x86_64	135
*updates	Fedora 30 - x86_64 - Updates	12,774
*updates-modular	Fedora Modular 30 - x86_64 - Updates	145

To add a repository, use `dnf config-manager --add_repo URL`, where URL is the full URL to the repository. To enable a repository, use `dnf config-manager --set-enabled REPO_ID`.

Likewise, to disable a repository use `dnf config-manager --set-disabled REPO_ID`. In both cases REPO_ID is the unique ID for the repository, which you can get using `dnf repolist`. Added repositories are enabled by default.

Repositories are stored in `.repo` files in the directory `/etc/yum.repos.d/`, with exactly the same syntax used for yum.

Zypper

zypper is the package management tool used on SUSE Linux and OpenSUSE. Feature-wise it is similar to apt and yum, being able to install, update and remove packages from a system, with automated dependency resolution.

Updating the Package Index

Like other package management tools, zypper works with repositories containing packages and metadata. This metadata needs to be refreshed from time to time, so that the utility will know about the latest packages available. To do a refresh, simply type:

```
# zypper refresh
```

Repository 'Non-OSS Repository' is up to date.

Repository 'Main Repository' is up to date.

Repository 'Main Update Repository' is up to date.

Repository 'Update Repository (Non-Oss)' is up to date.

All repositories have been refreshed.

zypper has an auto-refresh feature that can be enabled on a per-repository basis, meaning that some repos may be refreshed automatically before a query or package installation, and others may need to be refreshed manually. You will learn how to control this feature shortly.

Searching for Packages

To search for a package, use the search (or se) operator, followed by the package name:

```
# zypper se gnumeric
```

Loading repository data...

Reading installed packages...

S	Name	Summary	Type
--	+	+	+

```
| gnumeric      | Spreadsheet Application      | package
| gnumeric-devel | Spreadsheet Application      | package
| gnumeric-doc   | Documentation files for Gnumeric | package
| gnumeric-lang  | Translations for package gnumeric | package
```

The search operator can also be used to obtain a list of all the installed packages in the system. To do so, use the `-i` parameter without a package name, as in `zypper se -i`.

To see if a specific package is installed, add the package name to the command above. For example, the following command will search among the installed packages for any containing “firefox” in the name:

```
# zypper se -i firefox
Loading repository data...
Reading installed packages...
```

```
S | Name                | Summary                | Type
--+-----+-----+-----+
i | MozillaFirefox        | Mozilla Firefox Web B-> | package
i | MozillaFirefox-branding-openSUSE | openSUSE branding of -> | package
i | MozillaFirefox-translations-common | Common translations f-> | package
```

To search only among non-installed packages, add the `-u` parameter to the `se` operator.

Installing, Upgrading and Removing Packages

To install a software package, use the `install` (or `in`) operator, followed by the package name. Like so:

```
# zypper in unrar
zypper in unrar
Loading repository data...
Reading installed packages...
Resolving package dependencies...
```

The following NEW package is going to be installed:
unrar

```
1 new package to install.
Overall download size: 141.2 KiB. Already cached: 0 B. After the operation, additional 301.6 KiB will
be used.
Continue? [y/n/v/...? shows all options] (y): y
Retrieving package unrar-5.7.5-lp151.1.1.x86_64
(1/1), 141.2 KiB (301.6 KiB unpacked)
Retrieving: unrar-5.7.5-lp151.1.1.x86_64.rpm .....[done]
Checking for file conflicts: .....[done]
(1/1) Installing: unrar-5.7.5-lp151.1.1.x86_64 .....[done]
zypper can also be used to install an RPM package on disk, while trying to satisfy its dependencies
using packages from the repositories. To do so, just provide the full path to the package instead of a
package name, like zypper in /home/john/newpackage.rpm.
```

To update packages installed on the system, use `zypper update`. As in the installation process, this will show a list of packages to be installed/upgraded before asking if you want to proceed.

If you wish to only list the available updates, without installing anything, you can use `zypper list-updates`.

To remove a package, use the `remove` (or `rm`) operator, followed by the package name:

```
# zypper rm unrar
Loading repository data...
```

Reading installed packages...
Resolving package dependencies...

The following package is going to be REMOVED:
unrar

1 package to remove.

After the operation, 301.6 KiB will be freed.

Continue? [y/n/v/...? shows all options] (y): y

(1/1) Removing unrar-5.7.5-lp151.1.1.x86_64[done]

Keep in mind that removing a package also removes any other packages that depend on it. For example:

```
# zypper rm libgimp-2_0-0
```

Loading repository data...

Warning: No repositories defined. Operating only with the installed resolvables. Nothing can be installed.

Reading installed packages...

Resolving package dependencies...

The following 6 packages are going to be REMOVED:

gimp gimp-help gimp-lang gimp-plugins-python libgimp-2_0-0
libgimpui-2_0-0

6 packages to remove.

After the operation, 98.0 MiB will be freed.

Continue? [y/n/v/...? shows all options] (y):

Finding Which Packages Contain a Specific File

To see which packages contain a specific file, use the search operator followed by the --provides parameter and the name of the file (or full path to it). For example, if you want to know which packages contain the file libgimpmodule-2.0.so.0 in /usr/lib64/ you would use:

```
# zypper se --provides /usr/lib64/libgimpmodule-2.0.so.0
```

Loading repository data...

Reading installed packages...

S	Name	Summary	Type
---	------	---------	------

--+	-----	-----	-----
-----	-------	-------	-------

i	libgimp-2_0-0	The GNU Image Manipulation Program - Libra->	package
---	---------------	--	---------

Getting Package Information

To see the metadata associated with a package, use the info operator followed by the package name. This will provide you with the origin repository, package name, version, architecture, vendor, installed size, if it is installed or not, the status (if it is up-to-date), the source package and a description.

```
# zypper info gimp
```

Loading repository data...

Reading installed packages...

Information for package gimp:

Repository : Main Repository

Name : gimp

Version : 2.8.22-lp151.4.6

Arch : x86_64

Vendor : openSUSE

Installed Size : 29.1 MiB

Installed : Yes (automatically)
Status : up-to-date
Source package : gimp-2.8.22-lp151.4.6.src
Summary : The GNU Image Manipulation Program
Description :

The GIMP is an image composition and editing program, which can be used for creating logos and other graphics for Web pages. The GIMP offers many tools and filters, and provides a large image manipulation toolbox, including channel operations and layers, effects, subpixel imaging and antialiasing, and conversions, together with multilevel undo. The GIMP offers a scripting facility, but many of the included scripts rely on fonts that we cannot distribute.

Managing Software Repositories

zypper can also be used to manage software repositories. To see a list of all the repositories currently registered in your system, use `zypper repos`:

```
# zypper repos
```

Repository priorities are without effect. All enabled repositories share the same priority.

#	Alias	Name	Enabled	GPG Check	Refresh
1	openSUSE-Leap-15.1-1	openSUSE-Leap-15.1-1	No	----	----
2	repo-debug	Debug Repository	No	----	----
3	repo-debug-non-oss	Debug Repository (Non-OSS)	No	----	----
4	repo-debug-update	Update Repository (Debug)	No	----	----
5	repo-debug-update-non-oss	Update Repository (Debug, Non-OSS)	No	----	----
6	repo-non-oss	Non-OSS Repository	Yes	(r) Yes	Yes
7	repo-oss	Main Repository	Yes	(r) Yes	Yes
8	repo-source	Source Repository	No	----	----
9	repo-source-non-oss	Source Repository (Non-OSS)	No	----	----
10	repo-update	Main Update Repository	Yes	(r) Yes	Yes
11	repo-update-non-oss	Update Repository (Non-Oss)	Yes	(r) Yes	Yes

See in the Enabled column that some repositories are enabled, while others are not. You can change this with the `modifyrepo` operator, followed by the `-e` (enable) or `-d` (disable) parameter and the repository alias (the second column in the output above).

```
# zypper modifyrepo -d repo-non-oss
```

Repository 'repo-non-oss' has been successfully disabled.

```
# zypper modifyrepo -e repo-non-oss
```

Repository 'repo-non-oss' has been successfully enabled.

Previously we mentioned that zypper has an auto refresh capability that can be enabled on a per-repository basis. When enabled, this flag will make zypper run a refresh operation (the same as running `zypper refresh`) before working with the specified repo. This can be controlled with the `-f` and `-F` parameters of the `modifyrepo` operator:

```
# zypper modifyrepo -F repo-non-oss
```

Autorefresh has been disabled for repository 'repo-non-oss'.

```
# zypper modifyrepo -f repo-non-oss
```

Autorefresh has been enabled for repository 'repo-non-oss'.

Adding and Removing Repositories

To add a new software repository for zypper, use the `addrepo` operator followed by the repository URL and repository name, like below:

```
# zypper addrepo http://packman.inode.at/suse/openSUSE_Leap_15.1/ packman
Adding repository 'packman' .....[done]
```

Repository 'packman' successfully added

```
URI      : http://packman.inode.at/suse/openSUSE_Leap_15.1/
Enabled  : Yes
GPG Check : Yes
Autorefresh : No
Priority  : 99 (default priority)
```

Repository priorities are without effect. All enabled repositories share the same priority. While adding a repository, you can enable auto-updates with the `-f` parameter. Added repositories are enabled by default, but you can add and disable a repository at the same time by using the `-d` parameter.

To remove a repository, use the `removerepo` operator, followed by the repository name (Alias). To remove the repository added in the example above, the command would be:

```
# zypper removerepo packman
Removing repository 'packman' .....[done]
Repository 'packman' has been removed.
```

Linux as a virtualization guest

Introduction

One of the great strengths of Linux is its versatility. One aspect of this versatility is the ability to use Linux as a means of hosting other operating systems, or individual applications, in a completely isolated and secure environment. This lesson will focus on the concepts of virtualization and container technologies, along with some technical details that should be taken into consideration when deploying a virtual machine on a cloud platform.

Virtualization Overview

Virtualization is a technology that allows for a software platform, called a hypervisor, to run processes that contain a fully emulated computer system. The hypervisor is responsible for managing the physical hardware's resources that can be used by individual virtual machines. These virtual machines are called guests of the hypervisor. A virtual machine has many aspects of a physical computer emulated in software, such as a system's BIOS and hard drive disk controllers. A virtual machine will often use hard disk images that are stored as individual files, and will have access to the host machine's RAM and CPU through the hypervisor software. The hypervisor separates the access to the host system's hardware resources among the guests, thus allowing for multiple operating systems to run on a single host system.

Commonly used hypervisors for Linux include:

Xen

Xen is an open source Type-1 hypervisor, meaning that it does not rely on an underlying operating system to function. A hypervisor of this sort is known as a bare-metal hypervisor since the computer can boot directly into the hypervisor.

KVM

The Kernel Virtual Machine is a Linux kernel module for virtualization. KVM is a hypervisor of both Type-1 and Type-2 hypervisor because, although it needs a generic Linux operating system to work, it is able to perform as a hypervisor perfectly well by integrating with a running Linux installation. Virtual machines deployed with KVM use the `libvirt` daemon and associated software utilities to be created and managed.

VirtualBox

A popular desktop application that makes it easy to create and manage virtual machines. Oracle VM VirtualBox is cross-platform, and will work on Linux, macOS, and Microsoft Windows. Since VirtualBox requires an underlying operating system to run, it is a Type-2 hypervisor.

Some hypervisors allow for the dynamic relocation of a virtual machine. The process of moving a virtual machine from one hypervisor installation to another is called a migration, and the techniques involved differ between hypervisor implementations. Some migrations can only be performed when the guest system has been completely shut down, and other can be performed while the guest is running (called a live migration). Such techniques can be of aid during maintenance windows for hypervisors, or for system resiliency when a hypervisor becomes non-functional and the guest can be moved to a hypervisor that is working.

Types of Virtual Machines

There are three main types of virtual machines, the fully virtualized guest, the paravirtualized guest and the hybrid guest.

Fully Virtualized

All instructions that a guest operating system is expected to execute must be able to run within a fully virtualized operating system installation. The reason for this is that no additional software drivers are installed within the guest to translate the instructions to either simulated or real hardware. A fully virtualized guest is one where the guest (or HardwareVM) is unaware that it is a running virtual machine instance. In order for this type of virtualization to take place on x86 based hardware the Intel VT-x or AMD-V CPU extensions must be enabled on the system that has the hypervisor installed. This can be done from a BIOS or UEFI firmware configuration menu.

Paravirtualized

A paravirtualized guest (or PVM) is one where the guest operating system is aware that it is a running virtual machine instance. These types of guests will make use of a modified kernel and special drivers (known as guest drivers) that will help the guest operating system utilize software and hardware resources of the hypervisor. The performance of a paravirtualized guest is often better than that of the fully virtualized guest due to the advantage that these software drivers provide.

Hybrid

Paravirtualization and full virtualization can be combined to allow unmodified operating systems to receive near native I/O performance by using paravirtualized drivers on fully virtualized operating systems. The paravirtualized drivers contain storage and network device drivers with enhanced disk and network I/O performance.

Virtualization platforms often provide packaged guest drivers for virtualized operating systems. The KVM utilizes drivers from the Virtio project, whereas Oracle VM VirtualBox uses Guest Extensions available from a downloadable ISO CD-ROM image file.

Example libvirt Virtual Machine

We will look at an example virtual machine that is managed by libvirt and uses the KVM hypervisor. A virtual machine often consists of a group of files, primarily an XML file that defines the virtual machine (such as its hardware configuration, network connectivity, display capabilities, and more) and an associated hard disk image file that contains the installation of the operating system and its software.

First, let us start to examine an example XML configuration file for a virtual machine and its network

environment:

```
$ ls /etc/libvirt/qemu
total 24
drwxr-xr-x 3 root root 4096 Oct 29 17:48 networks
-rw----- 1 root root 5667 Jun 29 17:17 rhel8.0.xml
```

Note

The qemu portion of the directory path refers to the underlying software that KVM-based virtual machines rely on. The QEMU project provides software for the hypervisor to emulate hardware devices that the virtual machine will use, such as disk controllers, access to the host CPU, network card emulation, and more.

Note that there is a directory named `networks`. This directory contains definition files (also using XML) that create network configurations that the virtual machines can use. This hypervisor is only using one network, and so there is only one definition file that contains a configuration for a virtual network segment that these systems will utilize

Example Virtual Machine Disk Storage

This virtual machine's hard disk image resides at `/var/lib/libvirt/images/rhel8`. Here is the disk image itself on this hypervisor:

```
$ sudo ls -lh /var/lib/libvirt/images/rhel8
-rw----- 1 root root 5.5G Oct 25 15:57 /var/lib/libvirt/images/rhel8
```

The current size of this disk image consumes only 5.5 GB of space on the hypervisor. However, the operating system within this guest sees a 23.3 GB sized disk, as evidenced by the output of the following command from within the running virtual machine:

```
$ lsblk
NAME        MAJ:MIN RM  SIZE RO TYPE MOUNTPOINT
vda         252:0    0 23.3G  0 disk
├─vda1      252:1    0   1G  0 part /boot
└─vda2      252:2    0 22.3G  0 part
   ├─rhel-root 253:0    0  20G  0 lvm  /
   └─rhel-swap 253:1    0   2.3G  0 lvm  [SWAP]
```

This is due to the type of disk provisioning used for this guest. There are multiple types of disk images that a virtual machine can use but the two primary types are:

COW

Copy-on-write (also referred to as *thin-provisioning* or *sparse images*) is a method where a disk file is created with a pre-defined upper size limit. The disk image size only increases as new data is written to the disk. Just like the previous example, the guest operating system sees the predefined disk limit of 23.3 GB, but has only written 5.5 GB of data to the disk file. The disk image format used for the example virtual machine is `qcow2` which is a QEMU COW file format.

RAW

A *raw* or *full* disk type is a file that has all of its space pre-allocated. For example, a 10 GB raw disk image file consumes 10 GB of actual disk space on the hypervisor. There is a performance benefit to this style of disk as all of the needed disk space already exists, so the underlying hypervisor can just write data to the disk without the performance hit of monitoring the disk image to ensure that it has not yet reached its limit and extending the size of the file as new data is written to it.

There are other virtualization management platforms such as *Red Hat Enterprise Virtualization* and *oVirt* that can

make use of physical disks to act as backing storage locations for a virtual machine's operating system. These systems can utilize storage area network (SAN) or network attached storage (NAS) devices to write their data to, and the hypervisor keeps track of what storage locations belong to which virtual machines. These storage systems can use technologies such as logical volume management (LVM) to grow or shrink the size of a virtual machine's disk storage as needed, and to aid in the creation and management of storage snapshots.

Working with Virtual Machine Templates

Since virtual machines are typically just files running on a hypervisor, it is easy to create templates that can be customized for particular deployment scenarios. Often a virtual machine will have a basic operating system installation and some pre-configured authentication configuration settings set up to ease future system launches. This cuts down on the amount of time it takes to build a new system by reducing the amount of work that is often repeated, such as base package installation and locale settings.

This virtual machine template could then later get copied to a new guest system. In this case, the new guest would get renamed, a new MAC address generated for its network interface, and other modifications can be made depending on its intended use.

The D-Bus Machine ID

Many Linux installations will utilize a machine identification number generated at install time, called the D-Bus machine ID. However, if a virtual machine is cloned to be used as a template for other virtual machine installations, a new D-Bus machine ID would need to be created to ensure that system resources from the hypervisor get directed to the appropriate guest system.

The following command can be used to validate that a D-Bus machine ID exists for the running system:

```
$ dbus-uuidgen --ensure
```

If no error messages are displayed, then an ID exists for the system. To view the current D-Bus machine ID, run the following:

```
$ dbus-uuidgen --get  
17f2e0698e844e31b12ccd3f9aa4d94a
```

The string of text that is displayed is the current ID number. No two Linux systems running on a hypervisor should have the same D-Bus machine ID.

The D-Bus machine ID is located at `/var/lib/dbus/machine-id` and is symbolically linked to `/etc/machine-id`. Changing this ID number on a running system is discouraged as system instability and crashes are likely to occur. If two virtual machines do have the same D-Bus machine ID, follow the procedure below to generate a new one:

```
$ sudo rm -f /etc/machine-id  
$ sudo dbus-uuidgen --ensure=/etc/machine-id
```

In the event that `/var/lib/dbus/machine-id` is not a symbolic link back to `/etc/machine-id`, then `/var/lib/dbus/machine-id` will need to be removed.

Deploying Virtual Machines to the Cloud

There are a multitude of IaaS (infrastructure as a service) providers available that run hypervisor systems and that can deploy virtual guest images for an organization. Practically all of these providers have tools in place that allows an administrator to build, deploy and configure custom virtual machines based on a variety of Linux distributions. Many of these companies also have systems in place that allow for the deployment and migrations of virtual machines built from within a customer's organization.

When assessing a deployment of a Linux system in an IaaS environment, there are some key elements that an administrator should be cognizant of:

Computing Instances

Many cloud providers will charge usage rates based on "computing instances", or how much CPU time your cloud-based infrastructure will use. Careful planning of how much processing time applications will actually require will aid in keeping the costs of a cloud solution manageable.

Computing instances will also often refer to the number of virtual machines that are provisioned in a cloud environment. Again, the more instances of systems that are running at one time will also factor into how much overall CPU time an organization will be charged for.

Block Storage

Cloud providers also have various levels of block storage available for an organization to use. Some offerings are simply meant to be web-based network storage for files, and other offerings relate to external storage for a cloud provisioned virtual machine to use for hosting files.

The cost for such offerings will vary based on the amount of storage used, and the speed of the storage within the provider's data centers. Faster storage access typically will cost more, and conversely data "at rest" (as in archival storage) is often very inexpensive.

Networking

One of the main components of working with a cloud solutions provider is how the virtual network will be configured. Many IaaS providers will have some form of web-based utilities that can be utilized for the design and implementation of different network routes, subnetting, and firewall configurations. Some will even provide DNS solutions so that publicly accessible FQDN (fully qualified domain names) can be assigned to your internet facing systems. There are even "hybrid" solutions available that can connect an existing, on-premise network infrastructure to a cloud-based infrastructure through the means of a VPN (virtual private network), thus tying the two infrastructures together.

Securely Accessing Guests in the Cloud

The most prevalent method in use for accessing a remote virtual guest on a cloud platform is through the use of OpenSSH software. A Linux system that resides in the cloud would have the OpenSSH server running, while an administrator would use an OpenSSH client with pre-shared keys for remote access.

An administrator would run the following command

```
$ ssh-keygen
```

and follow the prompts to create a public and private SSH key pair. The private key remains on the administrator's local system (stored in `~/.ssh/`) and the public key gets copied to the remote cloud system, the exact same

method one would use when working with networked machines on a corporate LAN.

The administrator would then run the following command:

```
$ ssh-copy-id -i <public_key> user@cloud_server
```

This will copy the public SSH key from the key pair just generated to the remote cloud server. The public key will be recorded in the `~/.ssh/authorized_keys` file of the cloud server, and set the appropriate permissions on the file.

Note	If there is only one public key file in the <code>~/.ssh/</code> directory, then the <code>-i</code> switch can be omitted, as the <code>ssh-copy-id</code> command will default to the public key file in the directory (typically the file ending with the <code>.pub</code> extension).
-------------	---

Some cloud providers will automatically generate a key pair when a new Linux system is provisioned. The administrator will then need to download the private key for the new system from the cloud provider and store it on their local system. Note that the permissions for SSH keys must be `0600` for a private key, and `0644` for a public key.

Preconfiguring Cloud Systems

A useful tool that simplifies the deployments of cloud-based virtual machine is the cloud-init utility. This command, along with the associated configuration files and pre-defined virtual machine image, is a vendor-neutral method for deploying a Linux guest across a plethora of IaaS providers. Utilizing YAML (YAML Ain't Markup Language) plain-text files an administrator can pre-configure network settings, software package selections, SSH key configuration, user account creations, locale settings, along with a myriad of other options to quickly build out new systems.

During a new system's initial boot up, cloud-init will read in the settings from YAML configuration files and apply them. This process only needs to apply to a system's initial setup, and makes deploying a fleet of new systems on a cloud provider's platform easy.

The YAML file syntax used with cloud-init is called cloud-config. Here is a sample cloud-config file:

```
#cloud-config
timezone: Africa/Dar_es_Salaam
hostname: test-system
```

```
# Update the system when it first boots up
apt_update: true
apt_upgrade: true
```

```
# Install the Nginx web server
packages:
- nginx
```

Note that on the top line there is no space between the hash symbol (#) and the term cloud-config.

Note

cloud-init is not just for virtual machines. The cloud-init tool suite can also be used to pre-configure containers (such as LXD Linux containers) prior to deployment.

Containers

Container technology is similar in some aspects to a virtual machine, where you get an isolated environment to easily deploy an application. Whereas with a virtual machine an entire computer is emulated, a container uses just enough software to run an application. In this way, there is far less overhead.

Containers allow for greater flexibility over that of a virtual machine. An application container can be migrated from one host to another, just as a virtual machine can be migrated from one hypervisor to another. However, sometimes a virtual machine will need to be powered off before it could be migrated, whereas with a container the application is always running while it is getting migrated. Containers also make it easy to deploy new versions of applications in tandem with an existing version. As users close their sessions with running containers, these containers can get automatically removed from the system by the container orchestration software and replaced with the new version thus reducing downtime.

Note

There are numerous container technologies available for Linux, such as Docker, Kubernetes, LXD/LXC, systemd-nspawn, OpenShift and more. The exact implementation of a container software package is beyond the scope of the LPIC-1 exam.

Containers make use of the control groups (better known as cgroups) mechanism within the Linux kernel. The cgroup is a way to partition system resources such as memory, processor time as well as disk and network bandwidth for an individual application. An administrator can use cgroups directly to set system resource limits on an application, or a group of applications that could exist within a single cgroup. In essence this is what container software does for the administrator, along with providing tools that ease the management and deployment of cgroups.

Note

Currently, knowledge of cgroups are not necessary for passing the LPIC-1 exam. The concept of the cgroup is mentioned here so that the candidate would at least have some background knowledge of how an application is segregated for the sake of system resource utilization.

Work on the command line

Reference to LPI objectives

LPIC-1 version 5.0, Exam 101, Objective 103.1

Weight

4

Key knowledge areas

Use single shell commands and one line command sequences to perform basic tasks on the command line.

Use and modify the shell environment including defining, referencing and exporting environment variables.

Use and edit command history.

Invoke commands inside and outside the defined path.

Partial list of the used files, terms and utilities

bash

echo

env

export

pwd

set

unset

type

which

man

uname

history

.bash_history

Quoting

Introduction

Newcomers to the world of Linux administration and the Bash shell often feel a bit lost without the reassuring comforts of a GUI interface. They are used to having right-click access to the visual cues and contextual information that graphic file manager utilities make available. So it is important to quickly learn and master the relatively small set of command line tools through which you can

instantly tap into all the data offered by your old GUI — and more.

Getting System Information

While staring at the flashing rectangle of a command line prompt, your first question will probably be “Where am I?” Or, more precisely, “Where in the Linux filesystem am I right now and if, say, I created a new file, where would it live?” What you are after here is your present work directory, and the `pwd` command will tell you what you want to know:

```
$ pwd
/home/frank
```

Assuming that Frank is currently logged in to the system and he is now in his home directory: `/home/frank/`. Should Frank create an empty file using the `touch` command without specifying any other location in the filesystem, the file will be created within `/home/frank/`. Listing the directory contents using `ls` will show us that new file:

```
$ touch newfile
$ ls
newfile
```

Besides your location in the filesystem, you will often want information about the Linux system you are running. This might include the exact release number of your distribution or the Linux kernel version that is currently loaded. The `uname` tool is what you are after here. And, in particular, `uname` using the `-a` (“all”) option.

```
$ uname -a
Linux base 4.18.0-18-generic #19~18.04.1-Ubuntu SMP Fri Apr 5 10:22:13 UTC 2019 x86_64 x86_64
x86_64 GNU/Linux
```

Here, `uname` shows that Frank’s machine has the Linux kernel version 4.18.0 installed and is running Ubuntu 18.04 on a 64-bit (x86_64) CPU.

Getting Command Information

You will often come across documentation talking about Linux commands with which you are not yet familiar. The command line itself offers all kinds of helpful information on what commands do and how to effectively use them. Perhaps the most useful information is found within the many files of the man system.

As a rule, Linux developers write man files and distribute them along with the utilities they create. man files are highly structured documents whose contents are intuitively divided by standard section headings. Typing `man` followed by the name of a command will give you information that includes the command name, a brief usage synopsis, a more detailed description, and some important historical and licensing background. Here is an example:

```
$ man uname
NAME(1)      User Commands      UNAME(1)
NAME
  uname - print system information
SYNOPSIS
  uname [OPTION]...
DESCRIPTION
  Print certain system information. With no OPTION, same as -s.
  -a, --all
    print all information, in the following order, except omit -p
```

and -i if unknown:
-s, --kernel-name
 print the kernel name
-n, --nodename
 print the network node hostname
-r, --kernel-release
 print the kernel release
-v, --kernel-version
 print the kernel version
-m, --machine
 print the machine hardware name
-p, --processor
 print the processor type (non-portable)
-i, --hardware-platform
 print the hardware platform (non-portable)
-o, --operating-system
 print the operating system
--help display this help and exit
--version
 output version information and exit

AUTHOR

Written by David MacKenzie.

REPORTING BUGS

GNU coreutils online help: <<http://www.gnu.org/software/coreutils/>>

Report uname translation bugs to

<<http://translationproject.org/team/>>

COPYRIGHT

Copyright©2017 Free Software Foundation, Inc. License GPLv3+: GNU

GPL version 3 or later <<http://gnu.org/licenses/gpl.html>>.

This is free software: you are free to change and redistribute it.

There is NO WARRANTY, to the extent permitted by law.

SEE ALSO

arch(1), uname(2)

Full documentation at: <<http://www.gnu.org/software/coreutils/uname>>

or available locally via: info '(coreutils) uname invocation'

GNU coreutils 8.28 January 2018 UNAME(1)

man only works when you give it an exact command name. If, however, you are not sure about the name of the command you are after, you can use the apropos command to search through the man page names and descriptions. Assuming, for instance, that you cannot remember that it is uname that will give you your current Linux kernel version, you can pass the word kernel to apropos. You will probably get many lines of output, but they should include these:

```
$ apropos kernel
```

```
systemd-udevd-kernel.socket (8) - Device event managing daemon
```

```
uname (2)            - get name and information about current kernel
```

```
urandom (4)        - kernel random number source devices
```

If you do not need a command's full documentation, you can quickly get basic data about a command using type. This example uses type to query four separate commands at once. The results show us that cp ("copy") is a program that lives in /bin/cp and that kill (change the state of a running process) is a shell builtin — meaning that it is actually a part of the Bash shell itself:

```
$ type uname cp kill which
```

```
uname is hashed (/bin/uname)
```

```
cp is /bin/cp
```

```
kill is a shell builtin
```

```
which is /usr/bin/which
```

Notice that, besides being a regular binary command like cp, uname is also "hashed." That is

because Frank recently used `uname` and, to increase system efficiency, it was added to a hash table to make it more accessible the next time you run it. If he would run `type uname` after a system boot, Frank would find that `type` once again describes `uname` as a regular binary.

Note

A quicker way to clean up the hash table is to run the command `hash -d`.

Sometimes — particularly when working with automated scripts — you will need a simpler source of information about a command. The `which` command that our previous `type` command traced for us, will return nothing but the absolute location of a command. This example locates both the `uname` and `which` commands.

```
$ which uname which
/bin/uname
/usr/bin/which
```

Note

If you want to display information about “builtin” commands, you can use the `help` command.

Using Your Command History

You will often carefully research the proper usage for a command and successfully run it along with a complicated trail of options and arguments. But what happens a few weeks later when you need to run the same command with the same options and arguments but cannot remember the details? Rather than having to start your research over again from scratch, you will often be able to recover the original command using history.

Typing `history` will return the most recent commands you have executed with the most recent of those appearing last. You can easily search through those commands by piping a specific string to the `grep` command. This example will search for any command that included the text `bash_history`:

```
$ history | grep bash_history
1605 sudo find /home -name ".bash_history" | xargs grep sudo
Here a single command is returned along with its sequence number, 1605.
```

And speaking of `bash_history`, that is actually the name of a hidden file you should find within your user’s home directory. Since it is a hidden file (designated as such by the dot that precedes its filename), it will only be visible by listing the directory contents using `ls` with the `-a` argument:

```
$ ls /home/frank
newfile
$ ls -a /home/frank
. .. .bash_history .bash_logout .bashrc .profile .ssh newfile
```

What is in the `.bash_history` file? Take a look for yourself: you will see hundreds and hundreds of your most recent commands. You might, however, be surprised to find that some of your most recent commands are missing. That is because, while they are instantly added to the dynamic history database, the latest additions to your command history are not written to the `.bash_history` file until you exit your session.

You can leverage the contents of history to make your command line experience much faster and more efficient using the up and down arrow keys on your keyboard. Hitting the up key multiple times will populate the command line with recent commands. When you get to the one you would like to execute a second time, you can run it by pressing `Enter`. This makes it easy to recall and, if desired, modify commands multiple times during a shell session.

Finding Your Environment Variables

Introduction

An operating system environment includes the basic tools — like command line shells and sometimes a GUI — that you will need in order to get stuff done. But your environment will also come with a catalog of shortcuts and preset values. Here is where we will learn how to list, invoke, and manage those values.

Finding Your Environment Variables

So just how do we identify the current values for each of our environment variables? One way is through the `env` command:

```
$ env
DBUS_SESSION_BUS_ADDRESS=unix:path=/run/user/1000/bus
XDG_RUNTIME_DIR=/run/user/1000
XAUTHORITY=/run/user/1000/gdm/Xauthority
XDG_CONFIG_DIRS=/etc/xdg/xdg-ubuntu:/etc/xdg
PATH=/usr/local/sbin:/usr/local/bin:/usr/sbin:/usr/bin:/sbin:/bin:/usr/games:/usr/local/games:/snap/bin
GJS_DEBUG_TOPICS=JS ERROR;JS LOG
[...]
```

You will get a lot of output — much more than what is included in the above excerpt. But for now note the `PATH` entry, which contains the directories where your shell (and other programs) will look for other programs without having to specify a complete path. With that set, you could run a binary program that lives, say, in `/usr/local/bin` from within your home directory and it would run just as though the file was local.

Let us change the subject for a moment. The `echo` command will print to the screen whatever you tell it to. Believe it or not, there will be many times when getting `echo` to literally repeat something will be very useful.

```
$ echo "Hi. How are you?"
Hi. How are you?
```

But there is something else you can do with `echo`. When you feed it the name of an environment variable — and tell it that this is a variable by prefixing the variable name with a `$` — then, instead of just printing the variable's name, the shell will expand it giving you the value. Not sure whether your favorite directory is currently in the path? You can quickly check by running it through `echo`:

```
$ echo $PATH
/usr/local/sbin:/usr/local/bin:/usr/sbin:/usr/bin:/sbin:/bin:/usr/games:/usr/local/games:/snap/bin
```

Creating New Environment Variables

You can add your own custom variables to your environment. The simplest way is to use the `=` character. The string to the left will be the name of your new variable, and the string to the right will be its value. You can now feed the variable name to `echo` to confirm it worked:

```
$ myvar=hello
$ echo $myvar
hello
```

Note

Notice that there is no space on either side of the equal sign during variable assignment.

But did it really work? Type `bash` into the terminal to open a new shell. This new shell looks exactly like the one you were just in, but it is actually a child of the original one (which we call the parent). Now, inside this new child shell, try to get `echo` to do its magic the way it did before. Nothing. What's going on?

```
$ bash
$ echo $myvar
```

```
$
```

A variable you create the way we just did is only going to be available locally — within the immediate shell session. If you start up a new shell — or close down the session using `exit` — the variable will not go along with you. Typing `exit` here will take you back to your original parent shell which, right now, is where we want to be. You can run `echo $myvar` once again if you like just to confirm that the variable is still valid. Now type `export myvar` to pass the variable to any child shells that you may subsequently open. Try it out: type `bash` for a new shell and then `echo`:

```
$ exit
$ export myvar
$ bash
$ echo $myvar
hello
```

All this may feel a bit silly when we are creating shells for no real purpose. But understanding how shell variables are propagated through your system will become very important once you start writing serious scripts.

Deleting Environment Variables

Want to know how to clean up all those ephemeral variables you have created? One way is to simply close your parent shell — or reboot your computer. But there are simpler ways. Like, for instance, `unset`. Typing `unset` (without the `$`) will kill the variable. `echo` will now prove that.

```
$ unset myvar
$ echo $myvar
```

```
$
```

If there is an `unset` command, then you can bet there must be a `set` command to go with it. Running `set` by itself will display lots of output, but it is really not all that different from what `env` gave you. Look at the first line of output you will get when you filter for `PATH`:

```
$ set | grep PATH
PATH=/usr/local/sbin:/usr/local/bin:/usr/sbin:/usr/bin:/sbin:/bin:/usr/games:/usr/local/games:/snap/
bin
[...]
```

What is the difference between `set` and `env`? For our purposes, the main thing is that `set` will output all variables and functions. Let us illustrate that. We will create a new variable called `mynewvar` and then confirm it is there:

```
$ mynewvar=goodbye
$ echo $mynewvar
goodbye
```

Now, running `env` while using `grep` to filter for the string `mynewvar` will not display any output. But running `set` the same way will show us our local variable.

```
$ env | grep mynewvar
```

```
$ set | grep mynewvar  
mynewvar=goodbye
```

Quoting to Escape Special Characters

Now is as good a time as any other to introduce you to the problem of special characters. Alphanumeric characters (a-z and 0-9) will normally be read literally by Bash. If you try to create a new file called myfile you would just type touch followed by myfile and Bash will know what to do with it. But if you want to include a special character in your filename, you will need to do a bit more work.

To illustrate this, we will type touch and follow it by the title: my big file. The problem is that there are two spaces there between words which Bash will interpret. While, technically, you would not call a space a “character,” it is like one in the sense that Bash will not read it literally. If you list the contents of your current directory, rather than one file called my big file, you will see three files named, respectively, my, big, and file. That is because Bash thought you wanted to create multiple files whose names you were passing in a list:

```
$ touch my big file
```

```
$ ls  
my big file
```

The spaces will be interpreted the same way if you delete (rm) the three files all in one command:

```
$ rm my big file
```

Now let us try it the right way. Type touch and the three parts of your filename but this time enclose the name in quotation marks. This time it worked. Listing the directory contents will show you a single file with the proper name.

```
$ touch "my big file"
```

```
$ ls  
'my big file'
```

There are other ways to get the same effect. Single quotes, for instance, work just as well as double quotes. (Note that single quotes will preserve the literal value of all characters, while double quotes will preserve all characters except for \$, `, \ and, on certain cases, !.)

```
$ rm 'my big file'
```

Prepending each special character with the backslash will “escape” the specialness of the character and cause Bash to read it literally.

```
$ touch my\ big\ file
```

Process text streams using filters

Introduction

Dealing with text is a major part of every systems administrator’s job. Doug McIlroy, a member of the original Unix development team, summarized the Unix philosophy and said (among other important things): “Write programs to handle text streams, because that is a universal interface.” Linux is inspired by the Unix operating system and it firmly adopts its philosophy, so an administrator must expect lots of text manipulation tools within a Linux distribution.

A Quick Review on Redirections and Pipes

Also from the Unix philosophy:

Write programs that do one thing and do it well.

Write programs to work together.

One major way of making programs work together is through piping and redirections. Pretty much all of your text manipulation programs will get text from a standard input (stdin), output it to a standard output (stdout) and send eventual errors to a standard error output (stderr). Unless you specify otherwise, the standard input will be what you type on your keyboard (the program will read it after you press the Enter key). Similarly, the standard output and errors will be displayed in your terminal screen. Let us see how this works.

In your terminal, type `cat` and then hit the Enter key. Then type some random text.

```
$ cat
This is a test
This is a test
Hey!
Hey!
It is repeating everything I type!
It is repeating everything I type!
(I will hit ctrl+c so I will stop this nonsense)
(I will hit ctrl+c so I will stop this nonsense)
^C
```

For more information about the `cat` command (the term comes from “concatenate”) please refer to the man pages.

Note

If you are working on a really plain installation of a Linux server, some commands such as `info` and `less` might not be available. If this is the case, install these tools using the proper procedure in your system as described in the corresponding lessons.

As demonstrated above if you do not specify where `cat` should read from, it will read from the standard input (whatever you type) and output whatever it reads to your terminal window (its standard output).

Now try the following:

```
$ cat > mytextfile
This is a test
I hope cat is storing this to mytextfile as I redirected the output
I will hit ctrl+c now and check this
^C
```

```
$ cat mytextfile
This is a test
I hope cat is storing this to mytextfile as I redirected the output
I will hit ctrl+c now and check this
The > (greater than) tells cat to direct its output to the mytextfile file, not the standard output. Now try this:
```

```
$ cat mytextfile > mynewtextfile
$ cat mynewtextfile
This is a test
```

I hope cat is storing this to mytextfile as I redirected the output

I will hit ctrl+c now and check this

This has the effect of copying mytextfile to mynewtextfile. You can actually verify that these two files have the same content by performing a diff:

```
$ diff mynewtextfile mytextfile
```

As there is no output, the files are equal. Now try the append redirection operator (>>):

```
$ echo 'This is my new line' >> mynewtextfile
```

```
$ diff mynewtextfile mytextfile
```

```
4d3
```

```
< This is my new line
```

So far we have used redirections to create and manipulate files. We can also use pipes (represented by the symbol |) to redirect the output of one program to another program. Let us find the lines where the word “this” is found:

```
$ cat mytextfile | grep this
```

I hope cat is storing this to mytextfile as I redirected the output

I will hit ctrl+c now and check this

```
$ cat mytextfile | grep -i this
```

```
This is a test
```

I hope cat is storing this to mytextfile as I redirected the output

I will hit ctrl+c now and check this

Now we have piped the output of cat to another command: grep. Notice when we ignore the case (using the -i option) we get an extra line as a result.

Reading a Compressed File

Viewing a File in a Pager

You know cat concatenates a file to the standard output (once a file is provided after the command). The file /var/log/syslog is where your Linux system stores everything important going on in your system. Using the sudo command to elevate privileges so as to be able to read the /var/log/syslog file:

```
$ sudo cat /var/log/syslog
```

...you will see messages scrolling very fast within your terminal window. You can pipe the output to the program less so the results will be paginated. By using less you can use the arrow keys to navigate through the output and also use vi like commands to navigate and search throughout the text.

However, rather than pipe the cat command into a pagination program it is more pragmatic to just use the pagination program directly:

```
$ sudo less /var/log/syslog
```

... (output omitted for clarity)

Getting a Portion of a Text File

If only the start or end of a file needs to be reviewed, there are other methods available. The command `head` is used to read the first ten lines of a file by default, and the command `tail` is used to read the last ten lines of a file by default. Now try:

```
$ sudo head /var/log/syslog
Nov 12 08:04:30 hypatia rsyslogd: [origin software="rsyslogd" swVersion="8.1910.0" x-pid="811"
x-info="https://www.rsyslog.com"] rsyslogd was HUPed
Nov 12 08:04:30 hypatia systemd[1]: logrotate.service: Succeeded.
Nov 12 08:04:30 hypatia systemd[1]: Started Rotate log files.
Nov 12 08:04:30 hypatia vdr: [928] video directory scanner thread started (pid=882, tid=928,
prio=low)
Nov 12 08:04:30 hypatia vdr: [882] registered source parameters for 'A - ATSC'
Nov 12 08:04:30 hypatia vdr: [882] registered source parameters for 'C - DVB-C'
Nov 12 08:04:30 hypatia vdr: [882] registered source parameters for 'S - DVB-S'
Nov 12 08:04:30 hypatia vdr: [882] registered source parameters for 'T - DVB-T'
Nov 12 08:04:30 hypatia vdr[882]: vdr: no primary device found - using first device!
Nov 12 08:04:30 hypatia vdr: [929] epg data reader thread started (pid=882, tid=929, prio=high)
$ sudo tail /var/log/syslog
Nov 13 10:24:45 hypatia kernel: [ 8001.679238] mce: CPU7: Core temperature/speed normal
Nov 13 10:24:46 hypatia dbus-daemon[2023]: [session uid=1000 pid=2023] Activating via systemd:
service name='org.freedesktop.Tracker1.Miner.Extract' unit='tracker-extract.service' requested by '1.73' (uid=1000 pid=2425 comm="/usr/lib/tracker/tracker-miner-fs ")
Nov 13 10:24:46 hypatia systemd[2004]: Starting Tracker metadata extractor...
Nov 13 10:24:47 hypatia dbus-daemon[2023]: [session uid=1000 pid=2023] Successfully activated
service 'org.freedesktop.Tracker1.Miner.Extract'
Nov 13 10:24:47 hypatia systemd[2004]: Started Tracker metadata extractor.
Nov 13 10:24:54 hypatia kernel: [ 8010.462227] mce: CPU0: Core temperature above threshold, cpu
clock throttled (total events = 502907)
Nov 13 10:24:54 hypatia kernel: [ 8010.462228] mce: CPU4: Core temperature above threshold, cpu
clock throttled (total events = 502911)
Nov 13 10:24:54 hypatia kernel: [ 8010.469221] mce: CPU0: Core temperature/speed normal
Nov 13 10:24:54 hypatia kernel: [ 8010.469222] mce: CPU4: Core temperature/speed normal
Nov 13 10:25:03 hypatia systemd[2004]: tracker-extract.service: Succeeded.
To help illustrate the number of lines displayed, we can pipe the output of the head command to the
nl command, which will display the number of lines of text streamed into the command:
```

```
$ sudo head /var/log/syslog | nl
1  Nov 12 08:04:30 hypatia rsyslogd: [origin software="rsyslogd" swVersion="8.1910.0" x-
pid="811" x-info="https://www.rsyslog.com"] rsyslogd was HUPed
2  Nov 12 08:04:30 hypatia systemd[1]: logrotate.service: Succeeded.
3  Nov 12 08:04:30 hypatia systemd[1]: Started Rotate log files.
4  Nov 12 08:04:30 hypatia vdr: [928] video directory scanner thread started (pid=882, tid=928,
prio=low)
5  Nov 12 08:04:30 hypatia vdr: [882] registered source parameters for 'A - ATSC'
6  Nov 12 08:04:30 hypatia vdr: [882] registered source parameters for 'C - DVB-C'
7  Nov 12 08:04:30 hypatia vdr: [882] registered source parameters for 'S - DVB-S'
8  Nov 12 08:04:30 hypatia vdr: [882] registered source parameters for 'T - DVB-T'
9  Nov 12 08:04:30 hypatia vdr[882]: vdr: no primary device found - using first device!
10 Nov 12 08:04:30 hypatia vdr: [929] epg data reader thread started (pid=882, tid=929,
prio=high)
```

The Basics of sed, the Stream Editor

Let us take a look at the other files, terms and utilities that do not have cat in their names. We can do this by passing the -v option to grep, which instructs the command to output only the lines not containing cat:

```
$ zcat ftu.txt.gz | grep -v cat
```

```
cut
head
less
md5sum
nl
od
paste
sed
sha256sum
sha512sum
sort
split
tail
tr
uniq
wc
```

Most of what we can do with grep we can also do with sed — the stream editor for filtering and transforming text (as stated in the sed manual page). First we will recover our ftu.txt file by decompressing our gzip archive of the file:

```
$ gunzip ftu.txt.gz
```

```
$ ls ftu*
```

```
ftu.txt
```

Now, we can use sed to list only the lines containing the string cat:

```
$ sed -n /cat/p < ftu.txt
```

```
bzcat
cat
xzcat
zcat
```

We have used the less-than sign < to direct the contents of the file ftu.txt into into our sed command. The word enclosed between slashes (i.e. /cat/) is the term we are searching for. The -n option instructs sed to produce no output (unless the ones later instructed by the p command). Try running this same command without the -n option to see what happens. Then try this:

```
$ sed /cat/d < ftu.txt
```

```
cut
head
less
md5sum
nl
od
paste
sed
sha256sum
sha512sum
sort
split
tail
tr
uniq
```

wc

If we do not use the -n option, sed will print everything from the file except for what the d instructs sed to delete from its output.

A common use of sed is to find and replace text within a file. Suppose you want to change every occurrence of cat to dog. You can use sed to do this by supplying the s option to swap out each instance of the first term, cat, for the second term, dog:

```
$ sed s/cat/dog/ < ftu.txt
```

```
bzdog
dog
cut
head
less
md5sum
nl
od
paste
sed
sha256sum
sha512sum
sort
split
tail
tr
uniq
wc
xzdog
zdog
```

Rather than using a redirection operator (<) to pass the ftu.txt file into our sed command, we can just have the sed command operate on the file directly. We will try that next, while simultaneously creating a backup of the original file:

```
$ sed -i.backup s/cat/dog/ ftu.txt
```

```
$ ls ftu*
```

```
ftu.txt  ftu.txt.backup
```

The -i option will perform an in-place sed operation on your original file. If you do not use the .backup after the -i parameter, you would just have rewritten your original file. Whatever you use as text after the -i parameter will be the name the original file will be saved to prior to the modifications you asked sed to perform.

Ensuring Data Integrity

We have demonstrated how easy it is to manipulate files in Linux. There are times where you may wish to distribute a file to someone else, and you want to be sure that the recipient ends up with a true copy of the original file. A very common use of this technique is practiced when Linux distribution servers host downloadable CD or DVD images of their software along with files that contain the calculated checksum values of those disc images. Here is an example listing from a Debian download mirror:

[PARENTDIR] Parent Directory

-

[SUM] MD5SUMS

2019-09-08 17:46 274

[CRT]	MD5SUMS.sign	2019-09-08 17:52 833
[SUM]	SHA1SUMS	2019-09-08 17:46 306
[CRT]	SHA1SUMS.sign	2019-09-08 17:52 833
[SUM]	SHA256SUMS	2019-09-08 17:46 402
[CRT]	SHA256SUMS.sign	2019-09-08 17:52 833
[SUM]	SHA512SUMS	2019-09-08 17:46 658
[CRT]	SHA512SUMS.sign	2019-09-08 17:52 833
[ISO]	debian-10.1.0-amd64-netinst.iso	2019-09-08 04:37 335M
[ISO]	debian-10.1.0-amd64-xfce-CD-1.iso	2019-09-08 04:38 641M
[ISO]	debian-edu-10.1.0-amd64-netinst.iso	2019-09-08 04:38 405M
[ISO]	debian-mac-10.1.0-amd64-netinst.iso	2019-09-08 04:38 334M

In the listing above, the Debian installer image files are accompanied by text files that contain checksums of the files from the various algorithms (MD5, SHA1, SHA256 and SHA512).

Note

A checksum is a value derived from a mathematical computation, based on a cryptographic hash function, against a file. There are different types of cryptographic hash functions that vary in strength. The exam will expect you to be familiar with using md5sum, sha256sum and sha512sum.

Once you download a file (for example, the debian-10.1.0-amd64-netinst.iso image) you would then compare the checksum of the file that was downloaded against a checksum value that was provided for you.

Here is an example to illustrate the point. We will calculate the SHA256 value of the ftu.txt file using the sha256sum command:

```
$ sha256sum ftu.txt
```

```
345452304fc26999a715652543c352e5fc7ee0c1b9deac6f57542ec91daf261c ftu.txt
```

The long string of characters preceding the file name is the SHA256 checksum value of this text file. Let us create a file that contains that value, so that we can use it to verify the integrity of our original text file. We can do this with the same sha256sum command and redirect the output to a file:

```
$ sha256sum ftu.txt > sha256.txt
```

Now, to verify the ftu.txt file, we just use the same command and supply the filename that contains our checksum value along with the -c switch:

```
$ sha256sum -c sha256.txt
```

```
ftu.txt: OK
```

The value contained within the file matches the calculated SHA256 checksum for our ftu.txt file, just as we would expect. However, if the original file were modified (such as a few bytes lost during a file download, or someone had deliberately tampered with it) the value check would fail. In such cases we know that our file is bad or corrupted, and we can not trust the integrity of its contents. To prove the point, we will add some text at the end of the file:

```
$ echo "new entry" >> ftu.txt
```

Now we will make an attempt to verify the file's integrity:

```
$ sha256sum -c sha256.txt
```

```
ftu.txt: FAILED
```

```
sha256sum: WARNING: 1 computed checksum did NOT match
```

And we see that the checksum does not match what was expected for the file. Therefore, we could not trust the integrity of this file. We could attempt to download a new copy of a file, report the failure of the checksum to the sender of the file, or report it to a data center security team depending on the importance of the file.

Looking Deeper into Files

The octal dump (od) command is often used for debugging applications and various files. By itself, the od command will just list out a file's contents in octal format. We can use our ftu.txt file from earlier to practice with this command:

```
$ od ftu.txt
0000000 075142 060543 005164 060543 005164 072543 005164 062550
0000020 062141 066012 071545 005163 062155 071465 066565 067012
0000040 005154 062157 070012 071541 062564 071412 062145 071412
0000060 060550 032462 071466 066565 071412 060550 030465 071462
0000100 066565 071412 071157 005164 070163 064554 005164 060564
0000120 066151 072012 005162 067165 070551 073412 005143 075170
0000140 060543 005164 061572 072141 000012
0000151
```

The first column of output is the byte offset for each line of output. Since od prints out information in octal format by default, each line begins with the byte offset of eight bits, followed by eight columns, each containing the octal value of the data within that column.

Tip

Recall that a byte is 8 bits in length.

Should you need to view a file's contents in hexadecimal format, use the -x option:

```
$ od -x ftu.txt
0000000 7a62 6163 0a74 6163 0a74 7563 0a74 6568
0000020 6461 6c0a 7365 0a73 646d 7335 6d75 6e0a
0000040 0a6c 646f 700a 7361 6574 730a 6465 730a
0000060 6168 3532 7336 6d75 730a 6168 3135 7332
0000100 6d75 730a 726f 0a74 7073 696c 0a74 6174
0000120 6c69 740a 0a72 6e75 7169 770a 0a63 7a78
0000140 6163 0a74 637a 7461 000a
0000151
```

Now each of the eight columns after the byte offset are represented by their hexadecimal equivalents.

One handy use of the od command is for debugging scripts. For example, the od command can show us characters that are not normally seen that exist within a file, such as newline entries. We can do this with the -c option, so that instead of displaying the numerical notation for each byte, these column entries will instead be shown as their character equivalents:

```
$ od -c ftu.txt
0000000 b z c a t \n c a t \n c u t \n h e
0000020 a d \n l e s s \n m d 5 s u m \n n
0000040 l \n o d \n p a s t e \n s e d \n s
0000060 h a 2 5 6 s u m \n s h a 5 1 2 s
0000100 u m \n s o r t \n s p l i t \n t a
0000120 i l \n t r \n u n i q \n w c \n x z
0000140 c a t \n z c a t \n
0000151
```

All of the newline entries within the file are represented by the hidden \n characters. If you just want to view all of the characters within a file, and do not need to see the byte offset information, the byte offset column can be removed from the output like so:

```
$ od -An -c ftu.txt
 b z c a t \n c a t \n c u t \n h e
 a d \n l e s s \n m d 5 s u m \n n
 l \n o d \n p a s t e \n s e d \n s
 h a 2 5 6 s u m \n s h a 5 1 2 s
 u m \n s o r t \n s p l i t \n t a
 i l \n t r \n u n i q \n w c \n x z
 c a t \n z c a t \n
```

Perform basic file management

Using ls to List Files

The ls command is one of the most important command line tools you should learn in order to navigate the file system.

In its basic form, ls will list file and directory names only:

```
$ ls
Desktop Downloads emp_salary file1 Music Public Videos
Documents emp_name examples.desktop file2 Pictures Templates
```

When used with -l, referred to as “long listing” format, it shows file or directory permissions, owner, size, modified date, time and name:

```
$ ls -l
total 60
drwxr-xr-x 2 frank frank 4096 Apr 1 2018 Desktop
drwxr-xr-x 2 frank frank 4096 Apr 1 2018 Documents
drwxr-xr-x 2 frank frank 4096 Apr 1 2018 Downloads
-rw-r--r-- 1 frank frank 21 Sep 7 12:59 emp_name
-rw-r--r-- 1 frank frank 20 Sep 7 13:03 emp_salary
-rw-r--r-- 1 frank frank 8980 Apr 1 2018 examples.desktop
-rw-r--r-- 1 frank frank 10 Sep 1 2018 file1
-rw-r--r-- 1 frank frank 10 Sep 1 2018 file2
drwxr-xr-x 2 frank frank 4096 Apr 1 2018 Music
drwxr-xr-x 2 frank frank 4096 Apr 1 2018 Pictures
drwxr-xr-x 2 frank frank 4096 Apr 1 2018 Public
drwxr-xr-x 2 frank frank 4096 Apr 1 2018 Templates
drwxr-xr-x 2 frank frank 4096 Apr 1 2018 Videos
```

The first character in the output indicates the file type:

-
for a regular file.

d
for a directory.

c
for a special file.

To show the file sizes in a human readable format add the option -h:

```
$ ls -lh
total 60K
drwxr-xr-x 2 frank frank 4.0K Apr 1 2018 Desktop
drwxr-xr-x 2 frank frank 4.0K Apr 1 2018 Documents
drwxr-xr-x 2 frank frank 4.0K Apr 1 2018 Downloads
-rw-r--r-- 1 frank frank 21 Sep 7 12:59 emp_name
-rw-r--r-- 1 frank frank 20 Sep 7 13:03 emp_salary
-rw-r--r-- 1 frank frank 8.8K Apr 1 2018 examples.desktop
-rw-r--r-- 1 frank frank 10 Sep 1 2018 file1
-rw-r--r-- 1 frank frank 10 Sep 1 2018 file2
drwxr-xr-x 2 frank frank 4.0K Apr 1 2018 Music
drwxr-xr-x 2 frank frank 4.0K Apr 1 2018 Pictures
drwxr-xr-x 2 frank frank 4.0K Apr 1 2018 Public
drwxr-xr-x 2 frank frank 4.0K Apr 1 2018 Templates
drwxr-xr-x 2 frank frank 4.0K Apr 1 2018 Videos
```

To list all files including hidden files (those starting with .) use the option -a:

```
$ ls -a
.          .dbus file1 .profile
..         Desktop file2 Public
.bash_history .dmrc .gconf .sudo_as_admin_successful
Configuration files such as .bash_history which are by default hidden are now visible.
```

In general, the ls command syntax is given by:

ls OPTIONS FILE

Where OPTIONS are any of the options shown previously (to view all the possible options run man ls), and FILE is the name of the file or directory's details you wish to list.

Note

When FILE is not specified, the current directory is implied.

Creating, Copying, Moving and Deleting Files

Creating files with touch

The touch command is the easiest way to create new, empty files. You can also use it to change the timestamps (i.e., modification time) of existing files and directories. The syntax for using touch is:

```
touch OPTIONS FILE_NAME(S)
```

Without any options, touch would create new files for any file names that are supplied as arguments, provided that files with such names do not already exist. touch can create any number of files simultaneously:

```
$ touch file1 file2 file3
```

This would create three new empty files named file1, file2 and file3.

Several touch options are specifically designed to allow the user to change the timestamps for files. For example, the -a option changes only the access time, while the -m option changes only the modification time. The use of both options together changes the access and also the modification times to the current time:

```
$ touch -am file3
```

Copying Files with cp

As a Linux user, you will often copy files from one location to another. Whether it is moving a music file from one directory to another or a system file, use cp for all copy tasks:

```
$ cp file1 dir2
```

This command can be literally interpreted as copy file1 into directory dir2. The result is the presence of file1 inside dir2. For this command to be executed successfully file1 should be existent in the user's current directory. Otherwise, the system reports an error with the message No such file or directory.

```
$ cp dir1/file1 dir2
```

In this case, observe that the path to file1 is more explicit. The source path can be expressed either as a relative or absolute path. Relative paths are given in reference to a specific directory, while absolute paths are not given with a reference. Below we shall further clarify this notion.

For the moment, just observe that this command copies file1 into the directory dir2. The path to file1 is given with more detail since the user is currently not located in dir1.

```
$ cp /home/frank/Documents/file2 /home/frank/Documents/Backup
```

In this third case, file2 located at /home/frank/Documents is copied into the directory /home/frank/Documents/Backup. The source path provided here is absolute. In the two examples above, the source paths are relative. When a path starts with the character / it is an absolute path, otherwise it is a relative path.

The general syntax for cp is:

```
cp OPTIONS SOURCE DESTINATION
```

SOURCE is the file to copy and DESTINATION the directory into which the file would be copied. SOURCE and DESTINATION can be specified either as absolute or relative paths.

Moving Files with mv

Just like cp for copying, Linux provides a command for moving and renaming files. It is called mv.

The move operation is analogue to the cut and paste operation you generally perform through a Graphical User Interface (GUI).

If you wish to move a file into a new location, use mv in the following way:

```
mv FILENAME DESTINATION_DIRECTORY
```

Here is an example:

```
$ mv myfile.txt /home/frank/Documents
```

The result is that myfile.txt is moved into destination /home/frank/Documents.

To rename a file, mv is used in the following way:

```
$ mv old_file_name new_file_name
```

This changes the name of the file from old_file_name to new_file_name.

By default, mv would not seek your confirmation (technically said "would not prompt") if you wish to overwrite (rename) an existing file. However, you can allow the system to prompt, by using the option -i:

```
$ mv -i old_file_name new_file_name
```

```
mv: overwrite 'new_file_name'?
```

This command would ask the user's permission before overwriting old_file_name to new_file_name.

Conversely, using the -f:

```
$ mv -f old_file_name new_file_name
```

would forcefully overwrite the file, without asking any permission.

Deleting Files with rm

rm is used to delete files. Think of it as an abbreviated form of the word “remove”. Note that the action of removing a file is usually irreversible thus this command should be used with caution.

```
$ rm file1
```

This command would delete file1.

```
$ rm -i file1
```

rm: remove regular file 'file1'?
This command would request the user for confirmation before deleting file1. Remember, we saw the -i option when using mv above.

```
$ rm -f file1
```

This command forcefully deletes file1 without seeking your confirmation.

Multiple files can be deleted at the same time:

```
$ rm file1 file2 file3
```

In this example file1, file2 and file3 are deleted simultaneously.

The syntax for rm is generally given by:

```
rm OPTIONS FILE
```

Creating and Deleting Directories

Creating Directories with mkdir

Creating directories is critical to organizing your files and folders. Files may be grouped together in a logical way by keeping them inside a directory. To create a directory, use mkdir:

```
mkdir OPTIONS DIRECTORY_NAME
```

where DIRECTORY_NAME is the name of the directory to be created. Any number of directories can be created simultaneously:

```
$ mkdir dir1
```

would create the directory dir1 in the user’s current directory.

```
$ mkdir dir1 dir2 dir3
```

The preceding command would create three directories dir1, dir2 and dir3 at the same time.

To create a directory together with its subdirectories use the option -p (“parents”):

```
$ mkdir -p parents/children
```

This command would create the directory structure parents/children, i.e. it would create the

directories parents and children. children would be located inside parents.

Removing Directories with rmdir

rmdir deletes a directory if it is empty. Its syntax is given by:

rmdir OPTIONS DIRECTORY

where DIRECTORY could be a single argument or a list of arguments.

\$ rmdir dir1

This command would delete dir1.

\$ rmdir dir1 dir2

This command would simultaneously delete dir1 and dir2.

You may remove a directory with its subdirectory:

\$ rmdir -p parents/children

This would remove the directory structure parents/children. Note that if any of the directories are not empty, they will not be deleted.

Recursive Manipulation of Files and Directories

To manipulate a directory and its contents, you need to apply recursion. Recursion means, do an action and repeat that action all down the directory tree. In Linux, the options -r or -R or --recursive are generally associated with recursion.

The following scenario would help you better understand recursion:

You list the contents of a directory students, which contains two subdirectories level 1 and level 2 and the file named frank. By applying recursion, the ls command would list the content of students i.e. level 1, level 2 and frank, but would not end there. It would equally enter subdirectories level 1 and level 2 and list their contents and so on down the directory tree.

Recursive Listing with ls -R

ls -R is used to list the contents of a directory together with its subdirectories and files.

\$ ls -R mydirectory

mydirectory/:

file1 newdirectory

mydirectory/newdirectory:

In the listing above, mydirectory including all its content are listed. You can observe mydirectory contains the subdirectory newdirectory and the file file1. newdirectory is empty that is why no content is shown.

In general, to list the contents of a directory including its subdirectories, use:

ls -R DIRECTORY_NAME

Adding a trailing slash to DIRECTORY_NAME has no effect:

\$ ls -R animal

is similar to

```
$ ls -R animal/
```

Recursive Copy with cp -r

cp -r (or -R or --recursive) allows you to copy a directory together with its all subdirectories and files.

```
$ tree mydir
```

```
mydir
|_file1
|_newdir
    |_file2
    |_insidenew
    |_lastdir
```

3 directories, 2 files

```
$ mkdir newcopy
```

```
$ cp mydir newcopy
```

cp: omitting directory 'mydir'

```
$ cp -r mydir newcopy
```

```
* tree newcopy
```

```
newcopy
|_mydir
    |_file1
    |_newdir
        |_file2
        |_insidenew
        |_lastdir
```

4 directories, 2 files

In the listing above, we observe that trying to copy mydir into newcopy, using cp without -r, the system displays the message cp: omitting directory 'mydir'. However, by adding the option -r all the contents of mydir including itself are copied into newcopy.

To copy directories and subdirectories use:

```
cp -r SOURCE DESTINATION
```

Recursive Deletion with rm -r

rm -r will remove a directory and all its contents (subdirectories and files).

Warning

Be very careful with either the -r or the option combination of -rf when used with the rm command. A recursive remove command on an important system directory could render the system unusable. Employ the recursive remove command only when absolutely certain that the contents of a directory are safe to remove from a computer.

In trying to delete a directory without using -r the system would report an error:

```
$ rm newcopy/
```

```
rm: cannot remove 'newcopy/': Is a directory
```

```
$ rm -r newcopy/
```

You have to add -r as in the second command for the deletion to take effect.

Note

You may be wondering why we do not use rmdir in this case. There is a subtle difference between the two commands. rmdir would succeed in deleting only if the given directory is empty whereas rm -r can be used irrespective of whether this directory is empty or not.

Add the option `-i` to seek confirmation before the file is deleted:

```
$ rm -ri mydir/
```

rm: remove directory 'mydir/'?

The system prompts before trying to delete mydir.

File Globbing and Wildcards

File globbing is a feature provided by the Unix/Linux shell to represent multiple filenames by using special characters called wildcards. Wildcards are essentially symbols which may be used to substitute for one or more characters. They allow, for example, to show all files that start with the letter A or all files that end with the letters .conf.

Wildcards are very useful as they can be used with commands such as `cp`, `ls` or `rm`.

The following are some examples of file globbing:

```
rm *
```

Delete all files in current working directory.

```
ls l?st
```

List all files with names beginning with l followed by any single character and ending with st.

```
rmdir [a-z]*
```

Remove all directories whose name starts with a letter.

Types of Wildcards

There are three characters that can be used as wildcards in Linux:

`*` (asterisk)

which represents zero, one or more occurrences of any character.

`?` (question mark)

which represents a single occurrence of any character.

`[]` (bracketed characters)

which represents any occurrence of the character(s) enclosed in the square brackets. It is possible to use different types of characters whether numbers, letters, other special characters. For example, the expression `[0-9]` matches all digits.

The Asterisk

An asterisk (`*`) matches zero, one or more occurrences of any character.

For example:


```
$ find /home -name *.png
```

This would find all files that end with .png such as photo.png, cat.png, frank.png. The find command will be explored further in a following lesson.

Similarly:

```
$ ls lpic-*.txt
```

would list all text files that start with the characters lpic- followed by any number of characters and end with .txt, such as lpic-1.txt and lpic-2.txt.

The asterisk wildcard can be used to manipulate (copy, delete or move) all the contents of a directory:

```
$ cp -r animal/* forest
```

In this example, all the contents of animal is copied into forest.

In general to copy all the contents of a directory we use:

```
cp -r SOURCE_PATH/* DEST_PATH
```

where SOURCE_PATH can be omitted if we are already in the required directory.

The asterisk, just as any other wildcard, could be used repeatedly in the same command and at any position:

```
$ rm *ate*
```

Filenames prefixed with zero, one or more occurrence of any character, followed by the letters ate and ending with zero, one or more occurrence of any character will be removed.

The Question Mark

The question mark (?) matches a single occurrence of a character.

Consider the listing:

```
$ ls
```

```
last.txt  lest.txt  list.txt  third.txt  past.txt
```

To return only the files that start with l followed by any single character and the characters st.txt, we use the question mark (?) wildcard:

```
$ ls l?st.txt
```

```
last.txt  lest.txt  list.txt
```

Only the files last.txt, lest.txt and list.txt are displayed as they match the given criteria.

Similarly,

```
$ ls ??st.txt
```

```
last.txt  lest.txt  list.txt  past.txt
```

output files that are prefixed with any two characters followed by the text st.txt.

Bracketed Characters

The bracketed wildcards matches any occurrence of the character(s) enclosed in the square brackets:

```
$ ls l[aef]st.txt  
last.txt  lest.txt
```

This command would list all files starting with l followed by any of the characters in the set aef and ending with st.txt.

The square brackets could also take ranges:

```
$ ls l[a-z]st.txt  
last.txt  lest.txt  list.txt
```

This outputs all files with names starting with l followed by any lower case letter in the range a to z and ending with st.txt.

Multiple ranges could also be applied in the square brackets:

```
$ ls  
student-1A.txt student-2A.txt student-3.txt  
$ ls student-[0-9][A-Z].txt  
student-1A.txt student-2A.txt
```

The listing shows a school directory with a list of registered students. To list only those students whose registration numbers meet the following criteria:

begin with student-

followed by a number, and an uppercase character

and end with .txt

Combining Wildcards

Wildcards can be combined as in:

```
$ ls  
last.txt  lest.txt  list.txt  third.txt  past.txt  
$ ls [plf]?st*  
last.txt  lest.txt  list.txt  past.txt
```

The first wildcard component ([plf]) matches any of the characters p, l or f. The second wildcard component (?) matches any single character. The third wildcard component (*) matches zero, one or many occurrences of any character.

```
$ ls  
file1.txt file.txt file23.txt fom23.txt  
$ ls f*[0-9].txt  
file1.txt file23.txt fom23.txt
```

The previous command displays all files that begin with the letter f, followed by any set of letters, at least one occurrence of a digit and ends with .txt. Note that file.txt is not displayed as it does not match this criteria.

How to Find Files

As you use your machine, files progressively grow in number and size. Sometimes it becomes difficult to locate a particular file. Fortunately, Linux provides `find` to quickly search and locate files. `find` uses the following syntax:

`find STARTING_PATH OPTIONS EXPRESSION`

`STARTING_PATH`

defines the directory where the search begins.

`OPTIONS`

controls the behavior and adds specific criteria to optimize the search process.

`EXPRESSION`

defines the search query.

```
$ find . -name "myfile.txt"
```

```
./myfile.txt
```

The starting path in this case is the current directory. The option `-name` specifies that the search is based on the name of the file. `myfile.txt` is the name of the file to search. When using file globbing, be sure to include the expression in quotation marks:

```
$ find /home/frank -name "*.png"
```

```
/home/frank/Pictures/logo.png
```

```
/home/frank/screenshot.png
```

This command finds all files ending with `.png` starting from `/home/frank/` directory and beneath. If you do not understand the usage of the asterisk (`*`), it is covered in the previous lesson.

Using Criteria to Speed Search

Use `find` to locate files based on type, size or time. By specifying one or more options, the desired results are obtained in less time.

Switches to finding files based on type include:

`-type f`

file search.

`-type d`

directory search.

`-type l`

symbolic link search.

```
$ find . -type d -name "example"
```

This command finds all directories in the current directory and below, that have the name `example`.

Other criteria which could be used with `find` include:

`-name`

performs a search based on the given name.

-iname
searches based on the name, however, the case is not important (i.e. the test case myFile is similar to MYFILE).

-not
returns those results that do not match the test case.

-maxdepth N
searches the current directory as well as subdirectories N levels deep.

Locating Files by Modification Time

find also allows to filter a directory hierarchy based on when the file was modified:

```
$ sudo find / -name "*.conf" -mtime 7  
/etc/logrotate.conf
```

This command would search for all files in the entire file system (the starting path is the root directory, i.e. /) that end with the characters .conf and have been modified in the last seven days. This command would require elevated privileges to access directories starting at the base of the system's directory structure, hence the use of sudo here. The argument passed to mtime represents the number of days since the file was last modified.

Locating Files by Size

find can also locate files by size. For example, searching for files larger than 2G in /var:

```
$ sudo find /var -size +2G  
/var/lib/libvirt/images/debian10.qcow2  
/var/lib/libvirt/images/rhel8.qcow2
```

The -size option displays files of sizes corresponding to the argument passed. Some example arguments include:

-size 100b
files which are exactly 100 bytes.

-size +100k
files taller than 100 kilobytes.

-size -20M
files smaller than 20 megabytes.

-size +2G
files larger than 2 gigabytes.

Note

To find empty files we can use: find . -size 0b or find . -empty.

Acting on the Result Set

Once a search is done, it is possible to perform an action on the resulting set by using `-exec`:

```
$ find . -name "*.conf" -exec chmod 644 '{}' \;
```

This filters every object in the current directory (.) and below for file names ending with `.conf` and then executes the `chmod 644` command to modify file permissions on the results.

For now, do not bother with the meaning of `'{}' \;` as it will be discussed later.

Using grep to Filter for Files Based on Content

`grep` is used to search for the occurrence of a keyword.

Consider a situation where we are to find files based on content:

```
$ find . -type f -exec grep "lpi" '{}' \; -print
```

```
./bash_history
```

```
Alpine/M
```

```
helping/M
```

This would search every object in the current directory hierarchy (.) that is a file (`-type f`) and then executes the command `grep "lpi"` for every file that satisfies the conditions. The files that match these conditions are printed on the screen (`-print`). The curly braces (`{}`) are a placeholder for the find match results. The `{}` are enclosed in single quotes (`'`) to avoid passing `grep` files with names containing special characters. The `-exec` command is terminated with a semicolon (`;`), which should be escaped (`\;`) to avoid interpretation by the shell.

Adding the option `-delete` to the end of an expression would delete all files that match. This option should be used when you are certain that the results only match the files that you wish to delete.

In the example below, `find` locates all files in the hierarchy starting at the current directory then deletes all files that end with the characters `.bak`:

```
$ find . -name "*.bak" -delete
```

Archiving Files

The `tar` Command (Archiving and Compression)

The `tar` command, short for “tape archive(r)”, is used to create tar archives by converting a group of files into an archive. Archives are created so as to easily move or backup a group of files. Think of `tar` as a tool that creates a glue onto which files can be attached, grouped and easily moved.

`tar` also has the ability to extract tar archives, display a list of the files included in the archive as well as add additional files to an existing archive.

The `tar` command syntax is as follows:

```
tar [OPERATION_AND_OPTIONS] [ARCHIVE_NAME] [FILE_NAME(S)]  
OPERATION
```

Only one operation argument is allowed and required. The most frequently used operations are:

`--create (-c)`

Create a new tar archive.

`--extract (-x)`

Extract the entire archive or one or more files from an archive.

`--list (-t)`

Display a list of the files included in the archive.

OPTIONS

The most frequently used options are:

`--verbose (-v)`

Show the files being processed by the tar command.

`--file=archive-name (-f archive-name)`

Specifies the archive file name.

ARCHIVE_NAME

The name of the archive.

FILE_NAME(S)

A space-separated list of file names to be extracted. If not provided the entire archive is extracted.

Creating an Archive

Let's say we have a directory named `stuff` in the current directory and we want to save it to a file named `archive.tar`. We would run the following command:

```
$ tar -cvf archive.tar stuff
```

```
stuff/
```

```
stuff/service.conf
```

Here's what those switches actually mean:

`-c`

Create an archive.

`-v`

Display progress in the terminal while creating the archive, also known as "verbose" mode. The `-v` is always optional in these commands, but it is helpful.

`-f`

Allows to specify the filename of the archive.

In general to archive a single directory or a single file on Linux, we use:

```
tar -cvf NAME-OF-ARCHIVE.tar /PATH/TO/DIRECTORY-OR-FILE
```

Note

tar works recursively. It will perform the required action on every subsequent directory inside the directory specified.

To archive multiple directories at once, we list all the directories delimiting them by a space in the section `/PATH/TO/DIRECTORY-OR-FILE`:

```
$ tar -cvf archive.tar stuff1 stuff2
```

This would produce an archive of stuff1 and stuff2 in archive.tar

Extracting an Archive

We can extract an archive using tar:

```
$ tar -xvf archive.tar
```

```
stuff/
```

```
stuff/service.conf
```

This will extract the contents of archive.tar to the current directory.

This command is the same as the archive creation command used above, except the -x switch that replaces the -c switch.

To extract the contents of the archive to a specific directory we use -C:

```
$ tar -xvf archive.tar -C /tmp
```

This will extract the contents of archive.tar to the /tmp directory.

```
$ ls /tmp
```

```
stuff
```

Compressing with tar

The GNU tar command included with Linux distributions can create a .tar archive and then compress it with gzip or bzip2 compression in a single command:

```
$ tar -czvf name-of-archive.tar.gz stuff
```

This command would create a compressed file using the gzip algorithm (-z).

While gzip compression is most frequently used to create .tar.gz or .tgz files, tar also supports bzip2 compression. This allows the creation of bzip2 compressed files, often named .tar.bz2, .tar.bz or .tbz files.

To do so, we replace -z for gzip with -j for bzip2:

```
$ tar -cjvf name-of-archive.tar.bz stuff
```

To decompress the file, we replace -c with -x, where x stands for “extract”:

```
$ tar -xzf archive.tar.gz
```

gzip is faster, but it generally compresses a bit less, so you get a somewhat larger file. bzip2 is slower, but it compresses a bit more, so you get a somewhat smaller file. In general, though, gzip and bzip2 are practically the same thing and both will work similarly.

Alternatively we may apply gzip or bzip2 compression using gzip command for gzip compressions and the bzip command for bzip compressions. For example, to apply gzip compression, use:

```
gzip FILE-TO-COMPRESS
```

```
gzip
```

creates the compressed file with the same name but with a .gz ending.

```
gzip
```

removes the original files after creating the compressed file.

The bzip2 command works in a similar fashion.

To uncompress the files we use either gunzip or bunzip2 depending on the algorithm used to compress a file.

The cpio Command

cpio stands for “copy in, copy out”. It is used to process archive files such as *.cpio or *.tar files.

cpio performs the following operations:

Copying files to an archive.

Extracting files from an archive.

It takes the list of files from the standard input (mostly output from ls).

To create a cpio archive, we use:

```
$ ls | cpio -o > archive.cpio
```

The -o option instructs cpio to create an output. In this case, the output file created is archive.cpio. The ls command lists the contents of the current directory which are to be archived.

To extract the archive we use :

```
$ cpio -id < archive.cpio
```

The -i option is used to perform the extract. The -d option would create the destination folder. The character < represents standard input. The input file to be extracted is archive.cpio.

The dd Command

dd copies data from one location to another. The command line syntax of dd differs from many other Unix programs, it uses the syntax option=value for its command line options rather than the GNU standard -option value or --option=value formats:

```
$ dd if=oldfile of=newfile
```

This command would copy the content of oldfile into newfile, where if= is the input file and of= refers to the output file.

Note

The dd command typically will not output anything to the screen until the command has finished. By providing the status=progress option, the console will display the amount of work getting done by the command. For example: dd status=progress if=oldfile of=newfile.

dd is also used in changing data to upper/lower case or writing directly to block devices such as /dev/sdb:

```
$ dd if=oldfile of=newfile conv=ucase
```

This would copy all the contents of oldfile into newfile and capitalise all of the text.

The following command will backup the whole hard disk located at `/dev/sda` to a file named `backup.dd`:

```
$ dd if=/dev/sda of=backup.dd bs=4096
```

Use streams, pipes and redirects

Introduction

All computer programs follow the same general principle: data received from some source is transformed to generate an intelligible outcome. In Linux shell context, the data source can be a local file, a remote file, a device (like a keyboard), etc. The program's output is usually rendered on a screen, but is also common to store the output data in a local filesystem, send it to a remote device, play it through audio speakers, etc.

Operating systems inspired by Unix, like Linux, offer a great variety of input/output methods. In particular, the method of file descriptors allows to dynamically associate integer numbers with data channels, so that a process can reference them as its input/output data streams.

Standard Linux processes have three communication channels opened by default: the standard input channel (most times simply called `stdin`), the standard output channel (`stdout`) and the standard error channel (`stderr`). The numerical file descriptors assigned to these channels are 0 to `stdin`, 1 to `stdout` and 2 to `stderr`. Communication channels are also accessible through the special devices `/dev/stdin`, `/dev/stdout` and `/dev/stderr`.

These three standard communication channels allow programmers to write code that reads and writes data without worrying about the kind of media it's coming from or going to. For example, if a program needs a set of data as its input, it can just ask for data from the standard input and whatever is being used as the standard input will provide that data. Likewise, the simplest method a program can use to display its output is to write it in the standard output. In a standard shell session, the keyboard is defined as the `stdin` and the monitor screen is defined as the `stdout` and `stderr`.

The Bash shell has the ability to reassign the communication channels when loading a program. It allows, for example, to override the screen as the standard output and use a file in the local filesystem as `stdout`.

Redirects

The reassignment of a channel's file descriptor in the shell environment is called a redirect. A redirect is defined by a special character within the command line. For example, to redirect the standard output of a process to a file, the greater than symbol `>` is positioned at the end of the command and followed by the path to the file that will receive the redirected output:

```
$ cat /proc/cpuinfo >/tmp/cpu.txt
```

By default, only the content coming to `stdout` is redirected. That happens because the numerical value of the file descriptor should be specified just before the greater than symbol and, when not specified, Bash redirects the standard output. Therefore, using `>` is equivalent to use `1>` (the value of `stdout`'s file descriptor is 1).

To capture the content of `stderr`, the redirect `2>` should be used instead. Most command-line

programs send debug information and error messages to the standard error channel. It is possible, for example, to capture the error message triggered by an attempt to read a non-existent file:

```
$ cat /proc/cpu_info 2>/tmp/error.txt
$ cat /tmp/error.txt
cat: /proc/cpu_info: No such file or directory
```

Both stdout and stderr are redirected to the same target with `&>` or `>&`. It's important to not place any spaces beside the ampersand, otherwise Bash will take it as the instruction to run the process in background and not to perform the redirect.

The target must be a path to a writable file, like `/tmp/cpu.txt`, or a writable file descriptor. A file descriptor target is represented by an ampersand followed by the file descriptor's numerical value. For example, `1>&2` redirects stdout to stderr. To do the opposite, stderr to stdout, `2>&1` should be used instead.

Although not very useful, given that there is a shorter way to do the same task, it is possible to redirect stderr to stdout and then redirect it to a file. For example, a redirect to write both stderr and stdout to a file named `log.txt` can be written as `>log.txt 2>&1`. However, the main reason for redirecting stderr to stdout is to allow parsing of debug and error messages. It is possible to redirect the standard output of a program to the standard input of another program, but it is not possible to directly redirect the standard error to the standard input of another program. Thus, program's messages sent to stderr first need to be redirected to stdout in order to be read by another program's stdin.

To just discard the output of a command, its content can be redirected to the special file `/dev/null`. For example, `>log.txt 2>/dev/null` saves the contents of stdout in the file `log.txt` and discards the stderr. The file `/dev/null` is writable by any user but no data can be recovered from it, as it is not stored anywhere.

An error message is presented if the specified target is not writable (if the path points to a directory or a read-only file) and no modification to the target is made. However, an output redirect overwrites an existing writable target without any confirmation. Files are overwritten by output redirects unless Bash option `noclobber` is enabled, which can be done for the current session with the command `set -o noclobber` or `set -C`:

```
$ set -o noclobber
$ cat /proc/cpu_info 2>/tmp/error.txt
-bash: /tmp/error.txt: cannot overwrite existing file
```

To unset the `noclobber` option for the current session, run `set +o noclobber` or `set +C`. To make the `noclobber` option persistent, it must be included in the user's Bash profile or in the system-wide profile.

Even with the `noclobber` option enabled it is possible to append redirected data to existent content. This is accomplished with a redirection written with two greater than symbols `>>`:

```
$ cat /proc/cpu_info 2>>/tmp/error.txt
$ cat /tmp/error.txt
cat: /proc/cpu_info: No such file or directory
cat: /proc/cpu_info: No such file or directory
```

In the previous example, the new error message was appended to the existing one in file `/tmp/error.txt`. If the file does not exist yet, it will be created with the new data.

The data source of the standard input of a process can be reassigned as well. The less than symbol `<` is used to redirect the content of a file to the stdin of a process. In this case, data flows from right to left: the reassigned descriptor is assumed to be 0 at the left of the less than symbol and the data source (a path to a file) must be at the right of the less than symbol. The command `uniq`, like most command line utilities for processing text, accepts data sent to stdin by default:

```
$ uniq -c </tmp/error.txt
```

```
2 cat: /proc/cpu_info: No such file or directory
```

The `-c` option makes `uniq` display how many times a repeated line appears in the text. As the numeric value of the redirected file descriptor was suppressed, the example command is equivalent to `uniq -c 0</tmp/error.txt`. To use a file descriptor other than 0 in an input redirect only makes sense in specific contexts, because it is possible for a program to ask for data at file descriptors 3, 4, etc. Indeed, programs can use any integer above 2 as new file descriptors for data input/output. For example, the following C code reads data from file descriptor 3 and just replicates it to file descriptor 4:

Note

The program must handle such file descriptors correctly, otherwise it could attempt an invalid read or write operation and crash.

```
#include <stdio.h>
```

```
int main(int argc, char **argv){
    FILE *fd_3, *fd_4;
    // Open file descriptor 3
    fd_3 = fdopen(3, "r");
    // Open file descriptor 4
    fd_4 = fdopen(4, "w");
    // Read from file descriptor 3
    char buf[32];
    while ( fgets(buf, 32, fd_3) != NULL ){
        // Write to file descriptor 4
        fprintf(fd_4, "%s", buf);
    }
    // Close both file descriptors
    fclose(fd_3);
    fclose(fd_4);
}
```

To test it, save the sample code as `fd.c` and compile it with `gcc -o fd fd.c`. This program needs file descriptors 3 and 4 to be available so it can read and write to them. As an example, the previously created file `/tmp/error.txt` can be used as the source for file descriptor 3 and the file descriptor 4 can be redirected to `stdout`:

```
$ ./fd 3</tmp/error.txt 4>&1
```

```
cat: /proc/cpu_info: No such file or directory
```

```
cat: /proc/cpu_info: No such file or directory
```

From a programmer's perspective, using file descriptors avoids having to deal with option parsing and filesystem paths. The same file descriptor can even be used as input and output. In this case, the file descriptor is defined in the command line with both less than and greater than symbols, like in `3<>/tmp/error.txt`.

Pipes

Introduction

One aspect of the Unix philosophy states that each program should have a specific purpose and should not try to incorporate features outside its scope. But keeping things simple does not mean less elaborated results, because different programs can be chained together to produce a combined output. The vertical bar character `|`, also known as the pipe symbol, can be used to create a pipeline

connecting the output of a program directly into the input of another program, whereas command substitution allows to store the output of a program in a variable or use it directly as an argument to another command.

Pipes

Unlike redirects, with pipes data flows from left to right in the command line and the target is another process, not a filesystem path, file descriptor or Here document. The pipe character `|` tells the shell to start all distinct commands at the same time and to connect the output of the previous command to the input of the following command, left to right. For example, instead of using redirects, the content of the file `/proc/cpuinfo` sent to the standard output by `cat` can be piped to the stdin of `wc` with the following command:

```
$ cat /proc/cpuinfo | wc
 208 1184 6096
```

In the absence of a path to a file, `wc` counts the number of lines, words and characters it receives on its stdin, as is the case in the example. Many pipes can be present in a compound command. In the following example, two pipes are used:

```
$ cat /proc/cpuinfo | grep 'model name' | uniq
model name      : Intel(R) Xeon(R) CPU      X5355 @ 2.66GHz
```

The content of file `/proc/cpuinfo` produced by `cat /proc/cpuinfo` was piped to the command `grep 'model name'`, which then selects only the lines containing the term `model name`. The machine running the example has many CPUs, so there are repeated lines with `model name`. The last pipe connects `grep 'model name'` to `uniq`, which is responsible for skipping any line equal to the previous one.

Pipes can be combined with redirects in the same command line. The previous example can be rewritten to a simpler form:

```
$ grep 'model name' </proc/cpuinfo | uniq
model name      : Intel(R) Xeon(R) CPU      X5355 @ 2.66GHz
```

The input redirect for `grep` is not strictly necessary as `grep` accepts a file path as argument, but the example demonstrates how to build such combined commands.

Pipes and redirects are exclusive, that is, one source can be mapped to only one target. Yet, it is possible to redirect an output to a file and still see it on the screen with program `tee`. To do it, the first program sends its output to the stdin of `tee` and a file name is provided to the latter to store the data:

```
$ grep 'model name' </proc/cpuinfo | uniq | tee cpu_model.txt
model name      : Intel(R) Xeon(R) CPU      X5355 @ 2.66GHz
```

```
$ cat cpu_model.txt
```

```
model name      : Intel(R) Xeon(R) CPU      X5355 @ 2.66GHz
```

The output of the last program in the chain, generated by `uniq`, is displayed and stored in the file `cpu_model.txt`. To not overwrite the content of the provided file but to append data to it, the option `-a` must be provided to `tee`.

Only the standard output of a process is captured by a pipe. Let's say you must to go through a long compilation process on the screen and at the same time save both the standard output and the standard error to a file for later inspection. Assuming your current directory does not have a `Makefile`, the following command will output an error:

```
$ make | tee log.txt
make: *** No targets specified and no makefile found. Stop.
```

Although shown on the screen, the error message generated by `make` was not captured by `tee` and the file `log.txt` was created empty. A redirect needs to be done before a pipe can capture the `stderr`:

```
$ make 2>&1 | tee log.txt
```

```
make: *** No targets specified and no makefile found. Stop.
```

```
$ cat log.txt
```

```
make: *** No targets specified and no makefile found. Stop.
```

In this example the stderr of make was redirected to the stdout, so tee was able to capture it with a pipe, display it on the screen and save it in the file log.txt. In cases like this, it may be useful to save the error messages for later inspection

Command Substitution

Another method to capture the output of a command is command substitution. By placing a command inside backquotes, Bash replaces it with its standard output. The following example shows how to use the stdout of a program as an argument to another program:

```
$ mkdir `date +%Y-%m-%d`
```

```
$ ls
```

```
2019-09-05
```

The output of the program date, the current date formatted as year-month-day, was used as an argument to create a directory with mkdir. An identical result is obtained by using \$() instead of backquotes:

```
$ rmdir 2019-09-05
```

```
$ mkdir $(date +%Y-%m-%d)
```

```
$ ls
```

```
2019-09-05
```

The same method can be used to store the output of a command as a variable:

```
$ OS=`uname -o`
```

```
$ echo $OS
```

```
GNU/Linux
```

The command uname -o outputs the generic name of the current operating system, which was stored in the session variable OS. To assign the output of a command to a variable is very useful in scripts, making it possible to store and evaluate the data in many distinct ways.

Depending on the output generated by the replaced command, the builtin command substitution may not be appropriate. A more sophisticated method to use the output of a program as the argument of another program employs an intermediate called xargs. The program xargs uses the contents it receives via stdin to run a given command with the contents as its argument. The following example shows xargs running the program identify with arguments provided by program find:

```
$ find /usr/share/icons -name 'debian*' | xargs identify -format "%f: %wx%h\n"
```

```
debian-swirl.svg: 48x48
```

```
debian-swirl.png: 22x22
```

```
debian-swirl.png: 32x32
```

```
debian-swirl.png: 256x256
```

```
debian-swirl.png: 48x48
```

```
debian-swirl.png: 16x16
```

```
debian-swirl.png: 24x24
```

```
debian-swirl.svg: 48x48
```

The program identify is part of ImageMagick, a set of command-line tools to inspect, convert and edit most image file types. In the example, xargs took all paths listed by find and put them as arguments to identify, which then shows the information for each file formatted as required by the option -format. The files found by find in the example are images containing the distribution logo in a Debian filesystem. -format is a parameter to identify, not to xargs.

Option `-n 1` requires `xargs` to run the given command with only one argument at a time. In the example's case, instead of passing all paths found by `find` as a list of arguments to `identify`, using `xargs -n 1` would execute `command identify` for each path separately. Using `-n 2` would execute `identify` with two paths as arguments, `-n 3` with three paths as arguments and so on. Similarly, when `xargs` process multiline contents — as is the case with input provided by `find` — the option `-L` can be used to limit how many lines will be used as arguments per command execution.

Note

Using `xargs` with option `-n 1` or `-L 1` to process output generated by `find` may be unnecessary. Command `find` has the option `-exec` to run a given command for each search result item.

If the paths have space characters, it is important to run `find` with the option `-print0`. This option instructs `find` to use a null character between each entry so the list can be correctly parsed by `xargs` (the output was suppressed):

```
$ find . -name '*avi' -print0 -o -name '*mp4' -print0 -o -name '*mkv' -print0 | xargs -0 du | sort -n
```

The option `-0` tells `xargs` the null character should be used as the separator. That way the file paths given by `find` are correctly parsed even if they have blank or other special characters in it. The previous example shows how to use the command `du` to find out the disk usage of every file found and then sort the results by size. The output was suppressed for conciseness. Note that for each search criteria it is necessary to place the `-print0` option for `find`.

By default, `xargs` places the arguments of the executed command last. To change that behavior, the option `-l` should be used:

```
$ find . -mindepth 2 -name '*avi' -print0 -o -name '*mp4' -print0 -o -name '*mkv' -print0 | xargs -0 -l PATH mv PATH ./
```

In the last example, every file found by `find` is moved to the current directory. As the source path(s) must be informed to `mv` before the target path, a substitution term is given to the option `-l` of `xargs` which is then appropriately placed alongside `mv`. By using the null character as separator, it is not necessary to enclose the substitution term with quotes.

top and ps

When it comes to process monitoring, two invaluable tools are `top` and `ps`. Whilst the former produces output dynamically, the latter does it statically. In any case, both are excellent utilities to have a comprehensive view of all processes in the system.

Interacting with top

To invoke `top`, simply type `top`:

```
$ top
```

```
top - 11:10:29 up 2:21, 1 user, load average: 0,11, 0,20, 0,14
Tasks: 73 total, 1 running, 72 sleeping, 0 stopped, 0 zombie
%Cpu(s): 0,0 us, 0,3 sy, 0,0 ni, 99,7 id, 0,0 wa, 0,0 hi, 0,0 si, 0,0 st
KiB Mem : 1020332 total, 909492 free, 38796 used, 72044 buff/cache
KiB Swap: 1046524 total, 1046524 free, 0 used. 873264 avail Mem
```

PID	USER	PR	NI	VIRT	RES	SHR	S	%CPU	%MEM	TIME+	COMMAND
436	carol	20	0	42696	3624	3060	R	0,7	0,4	0:00.30	top
4	root	20	0	0	0	0	S	0,3	0,0	0:00.12	kworker/0:0
399	root	20	0	95204	6748	5780	S	0,3	0,7	0:00.22	sshd

```

1 root    20   0   56872  6596  5208 S  0,0  0,6  0:01.29 systemd
2 root    20   0     0     0     0 S  0,0  0,0  0:00.00 kthreadd
3 root    20   0     0     0     0 S  0,0  0,0  0:00.02 ksoftirqd/0
5 root     0 -20     0     0     0 S  0,0  0,0  0:00.00 kworker/0:0H
6 root    20   0     0     0     0 S  0,0  0,0  0:00.00 kworker/u2:0
7 root    20   0     0     0     0 S  0,0  0,0  0:00.08 rcu_sched
8 root    20   0     0     0     0 S  0,0  0,0  0:00.00 rcu_bh
9 root    rt   0     0     0     0 S  0,0  0,0  0:00.00 migration/0
10 root    0 -20     0     0     0 S  0,0  0,0  0:00.00 lru-add-drain
(...)

```

top allows the user some interaction. By default, the output is sorted by the percentage of CPU time used by each process in descending order. This behavior can be modified by pressing the following keys from within top:

M

Sort by memory usage.

N

Sort by process ID number.

T

Sort by running time.

P

Sort by percentage of CPU usage.

Tip

To switch between descending/ascending order just press R.

Other interesting keys to interact with top are:

? or h

Help.

k

Kill a process. top will ask for the PID of the process to be killed as well as for the signal to be sent (by default SIGTERM or 15).

r

Change the priority of a process (renice). top will ask you for the nice value. Possible values range from -20 through 19, but only the superuser (root) can set it to a value which is negative or lower than the current one.

u

List processes from a particular user (by default processes from all users are shown).

c

Show programs' absolute paths and differentiate between userspace processes and kernelspace processes (in square brackets).

V

Forest/hierarchy view of processes.

t and m

Change the look of CPU and memory readings respectively in a four-stage cycle: the first two presses show progress bars, the third hides the bar and the fourth brings it back.

W

Save configuration settings to ~/.toprc.

Tip

A fancier and more user-friendly version of top is htop. Another — perhaps more exhaustive — alternative is atop. If not already installed in your system, use your package manager to install them and give them a try.

An Explanation of the output of top

top output is divided into two areas: the summary area and the task area.

The Summary Area in top

The summary area is made up of the the five top rows and gives us the following information:

top - 11:10:29 up 2:21, 1 user, load average: 0,11, 0,20, 0,14

current time (in 24-hour format): 11:20:29

uptime (how much time the system has been up and running): up 2:21

number of users logged in and load average of the CPU for the last 1, 5 and 15 minutes, respectively: load average: 0,11, 0,20, 0,14

Tasks: 73 total, 1 running, 72 sleeping, 0 stopped, 0 zombie (information about the processes)

total number of processes in active mode: 73 total

running (those being executed): 1 running

sleeping (those waiting to resume execution): 72 sleeping

stopped (by a job control signal): 0 stopped

zombie (those which have completed execution but are still waiting for their parent process to remove them from the process table): 0 zombie

%Cpu(s): 0,0 us, 0,3 sy, 0,0 ni, 99,7 id, 0,0 wa, 0,0 hi, 0,0 si, 0,0 st (percentage of CPU time spent on)

user processes: 0,0 us

system/kernel processes: 0,4 sy

processes set to a nice value — the nicer the value, the lower the priority: 0,0 ni

nothing — idle CPU time: 99,7 id

processes waiting for I/O operations: 0,0 wa

processes serving hardware interrupts — peripherals sending the processor signals that require attention: 0,0 hi

processes serving software interrupts: 0,0 si

processes serving other virtual machine's tasks in a virtual environment, hence steal time: 0,0 st

KiB Mem : 1020332 total, 909492 free, 38796 used, 72044 buff/cache (memory information in kilobytes)

the total amount of memory: 1020332 total

unused memory: 909492 free

memory in use: 38796 used

the memory which is buffered and cached to avoid excessive disk access: 72044 buff/cache

Notice how the total is the sum of the other three values — free, used and buff/cache — (roughly 1 GB in our case).

KiB Swap: 1046524 total, 1046524 free, 0 used. 873264 avail Mem (swap information in kilobytes)

the total amount of swap space: 1046524 total

unused swap space: 1046524 free

swap space in use: 0 used

the amount of swap memory that can be allocated to processes without causing more swapping:
873264 avail Mem

The Task Area in top: Fields and Columns

Below the summary area there comes the task area, which includes a series of fields and columns reporting information about the running processes:

PID

Process identifier.

USER

User who issued the command that generated the process.

PR

Priority of process to the kernel.

NI

Nice value of process. Lower values have a higher priority than higher ones.

VIRT

Total amount of memory used by process (including Swap).

RES

RAM memory used by process.

SHR

Shared memory of the process with other processes.

S

Status of process. Values include: S (interruptible sleep — waiting for an event to finish), R (runnable — either executing or in the queue to be executed) or Z (zombie — terminated child processes whose data structures have not yet been removed from the process table).

%CPU

Percentage of CPU used by process.

%MEM

Percentage of RAM used by process, that is, the RES value expressed as a percentage.

TIME+

Total time of activity of process.

COMMAND

Name of command/program which generated the process.

Viewing processes statically: ps

As said above, ps shows a snapshot of processes. To see all processes with a terminal (tty), type ps a:

```
$ ps a
PID TTY    STAT  TIME COMMAND
386 tty1   Ss+   0:00 /sbin/agetty --noclear tty1 linux
424 tty7   Ssl+  0:00 /usr/lib/xorg/Xorg :0 -seat seat0 (...)
655 pts/0   Ss    0:00 -bash
1186 pts/0   R+    0:00 ps a
(...)
```

An Explanation of ps Option Syntax and Output

Concerning options, ps can accept three different styles: BSD, UNIX and GNU. Let us see how each of these styles would work when reporting information about a particular process ID:

BSD

Options do not follow any leading dash:

```
$ ps p 811
PID TTY    STAT  TIME COMMAND
811 pts/0    S     0:00 -su
```

UNIX

Options do follow a leading dash:

```
$ ps -p 811
PID TTY      TIME CMD
811 pts/0    00:00:00 bash
```

GNU

Options are followed by double leading dashes:

```
$ ps --pid 811
PID TTY      TIME CMD
811 pts/0    00:00:00 bash
```

In all three cases, ps reports information about the process whose PID is 811 — in this case, bash.

Similarly, you can use ps to search for the processes started by a particular user:

ps U carol (BSD)

ps -u carol (UNIX)

ps --user carol (GNU)

Let us check on the processes started by carol:

```
$ ps U carol
PID TTY    STAT  TIME COMMAND
811 pts/0    S     0:00 -su
898 pts/0    R+    0:00 ps U carol
```

She started two processes: `bash (-su)` and `ps (ps U carol)`. The `STAT` column tells us the state of the process (see below).

We can get the best out of `ps` by combining some of its options. A very useful command (producing an output similar to that of `top`) is `ps aux` (BSD style). In this case, processes from all shells (not only the current one) are shown. The meaning of the switches are the following:

`a`

Show processes that are attached to a `tty` or terminal.

`u`

Display user-oriented format.

`x`

Show processes that are not attached to a `tty` or terminal.

`$ ps aux`

USER	PID	%CPU	%MEM	VSZ	RSS	TTY	STAT	START	TIME	COMMAND
root	1	0.0	0.1	204504	6780	?	Ss	14:04	0:00	/sbin/init
root	2	0.0	0.0	0	0	?	S	14:04	0:00	[kthreadd]
root	3	0.0	0.0	0	0	?	S	14:04	0:00	[ksoftirqd/0]
root	5	0.0	0.0	0	0	?	S<	14:04	0:00	[kworker/0:0H]
root	7	0.0	0.0	0	0	?	S	14:04	0:00	[rcu_sched]
root	8	0.0	0.0	0	0	?	S	14:04	0:00	[rcu_bh]
root	9	0.0	0.0	0	0	?	S	14:04	0:00	[migration/0]

(...)

Let us explain the columns:

USER

Owner of process.

PID

Process identifier.

%CPU

Percentage of CPU used.

%MEM

Percentage of physical memory used.

VSZ

Virtual memory of process in KiB.

RSS

Non-swapped physical memory used by process in KiB.

TT

Terminal (`tty`) controlling the process.

STAT

Code representing the state of process. Apart from `S`, `R` and `Z` (that we saw when describing the output of `top`), other possible values include: `D` (uninterruptible sleep — usually waiting for I/O), `T` (stopped — normally by a control signal). Some extra modifier include: `<` (high-priority — not nice to other processes), `N` (low-priority — nice to other processes), or `+` (in the foreground process group).

STARTED

Time at which the process started.

TIME

Accumulated CPU time.

COMMAND

Command that started the process.

Create, monitor and kill processes

Job Control

Introduction

Every time we invoke a command, one or more processes are started. A well-trained system administrator not only needs to create processes, but also be able to keep track of them and send them different types of signals if and when required. In this lesson we will look at job control and how to monitor processes.

Job Control

Jobs are processes that have been started interactively through a terminal, sent to the background and have not yet finished execution. You can find out about the active jobs (and their status) in your Linux system by running jobs:

```
$ jobs
```

The jobs command above did not produce any output, which means there are no active jobs at the moment. Let us create our first job by running a command that takes some time to finish executing (the sleep command with a parameter of 60) and — while running — press Ctrl+Z:

```
$ sleep 60
```

```
^Z
```

```
[1]+  Stopped                  sleep 60
```

The execution of the command has been stopped (or — rather — suspended) and the command prompt is available again. You can look for jobs a second time and will find the suspended one now:

```
$ jobs
```

```
[1]+  Stopped                  sleep 60
```

Let us explain the output:

```
[1]
```

This number is the job ID and can be used — preceded by a percentage symbol (%) — to change the job status by the fg, bg and kill utilities (as you will be shown later).

```
+
```

The plus sign indicates the current, default job (that is, the last one being suspended or sent to the background). The previous job is flagged with a minus sign (-). Any other prior jobs are not flagged.

Stopped

Description of the job status.

```
sleep 60
```

The command or job itself.

With the -l option, jobs will additionally display the process ID (PID) right before the status:

```
$ jobs -l
[1]+ 1114 Stopped          sleep 60
The remaining possible options of jobs are:
```

-n
Lists only processes that have changed status since the last notification. Possible status include, Running, Stopped, Terminated or Done.

-p
Lists process IDs.

-r
Lists only running jobs.

-s
Lists only stopped (or suspended) jobs.

Note
Remember, a job has both a job ID and a process ID (PID).

Job Specification
The jobs command as well as other utilities such as fg, bg and kill (that you will see in the next section) need a job specification (or jobspec) to act upon a particular job. As we have just seen, this can be — and normally is — the job ID preceded by %. However, other job specifications are also possible. Let us have a look at them:

%n
Job whose id number is n:

```
$ jobs %1
[1]+  Stopped          sleep 60
%str
Job whose command line starts with str:
```

```
$ jobs %sl
[1]+  Stopped          sleep 60
%?str
Job whose command line contains str:
```

```
$ jobs %?le
[1]+  Stopped          sleep 60
%+ or %%
Current job (the one that was last started in the background or suspended from the foreground):
```

```
$ jobs %+
[1]+  Stopped          sleep 60
%-
Previous job (the one that was %+ before the default, current one):
```

```
$ jobs %-
[1]+  Stopped          sleep 60
In our case, since there is only one job, it is both current and previous.
```

Job Status: Suspension, Foreground and Background
Once a job is in the background or has been suspended, we can do any of three things to it:

Take it to the foreground with fg:

```
$ fg %1  
sleep 60
```

fg moves the specified job to the foreground and makes it the current job. Now we can wait until it finishes, stop it again with Ctrl+Z or terminate it with Ctrl+C.

Take it to the background with bg:

```
$ bg %1  
[1]+ sleep 60 &
```

Once in the background, the job can be brought back into the foreground with fg or killed (see below). Note the ampersand (&) meaning the job has been sent to the background. As a matter of fact, you can also use the ampersand to start a process directly in the background:

```
$ sleep 100 &  
[2] 970
```

Together with the job ID of the new job ([2]), we now get its process ID (970) too. Now both jobs are running in the background:

```
$ jobs  
[1]- Running      sleep 60 &  
[2]+ Running      sleep 100 &
```

A bit later the first job finishes execution:

```
$ jobs  
[1]- Done         sleep 60  
[2]+ Running      sleep 100 &
```

Terminate it through a SIGTERM signal with kill:

```
$ kill %2
```

To make sure the job has been terminated, run jobs again:

```
$ jobs  
[2]+ Terminated  sleep 100
```

Note

If no job is specified, fg and bg will act upon the current, default one. kill, however, always needs a job specification.

Detached Jobs: nohup

The jobs we have seen in the previous sections were all attached to the session of the user who invoked them. That means that if the session is terminated, the jobs are gone. However, it is possible to detach jobs from sessions and have them run even after the session is closed. This is achieved with the nohup (“no hangup”) command. The syntax is as follows:

```
nohup COMMAND &
```

Remember, the & sends the process into the background and frees up the terminal you are working at.

Let us detach the background job ping localhost from the current session:

```
$ nohup ping localhost &  
[1] 1251  
$ nohup: ignoring input and appending output to 'nohup.out'  
^C
```

The output shows us the job ID ([1]) and the PID (1251), followed by a message telling us about the

file nohup.out. This is the default file where stdout and stderr will be saved. Now we can press Ctrl+C to free up the command prompt, close the session, start another one as root and use tail -f to check if the command is running and output is being written to the default file:

```
$ exit
logout
$ tail -f /home/carol/nohup.out
64 bytes from localhost (::1): icmp_seq=3 ttl=64 time=0.070 ms
64 bytes from localhost (::1): icmp_seq=4 ttl=64 time=0.068 ms
64 bytes from localhost (::1): icmp_seq=5 ttl=64 time=0.070 ms
^C
```

Tip

Instead of using the default nohup.out you could have specified the output file of your choice with nohup ping localhost > /path/to/your/file &.

If we want to kill the process, we should specify its PID:

```
# kill 1251
```

Process Monitoring

A process or task is an instance of a running program. Thus, you create new processes every time you type in commands into the terminal.

The watch command executes a program periodically (2 seconds by default) and allows us to watch the program's output change over time. For instance, we can monitor how the load average changes as more processes are run by typing watch uptime:

```
Every 2.0s: uptime      debian: Tue Aug 20 23:31:27 2019
```

```
23:31:27 up 21 min, 1 user, load average: 0.00, 0.00, 0.00
```

The command runs until interrupted so we would have to stop it with Ctrl+C. We get two lines as output: the first one corresponds to watch and tells us how often the command will be run (Every 2.0s: uptime), what command/program to watch (uptime) as well as the hostname and date (debian: Tue Aug 20 23:31:27 2019). The second line of output is uptime's and includes the time (23:31:27), how much time the system has been up (up 21 min), the number of active users (1 user) and the average system load or number of processes in execution or in waiting state for the last 1, 5 and 15 minutes (load average: 0.00, 0.00, 0.00).

Similarly, you can check on memory use as new processes are created with watch free:

```
Every 2.0s: free        debian: Tue Aug 20 23:43:37 2019
```

```
23:43:37 up 24 min, 1 user, load average: 0.00, 0.00, 0.00
```

	total	used	free	shared	buff/cache	available
Mem:	16274868	493984	14729396	35064	1051488	15462040
Swap:	16777212	0	16777212			

To change the update interval for watch use the -n or --interval options plus the number of seconds as in:

```
$ watch -n 5 free
```

Now the free command will run every 5 seconds.

For more information about the options for uptime, free and watch, please refer to their manual pages.

Note

The information provided by uptime and free is also integrated in the more comprehensive tools top and ps (see below).

Sending Signals to Processes: kill

Every single process has a unique process identifier or PID. One way of finding out the PID of a process is by using the pgrep command followed by the process' name:

```
$ pgrep sleep
1201
```

Note

A process' identifier can also be discovered through the pidof command (e.g. pidof sleep).

Similar to pgrep, pkill command kills a process based on its name:

```
$ pkill sleep
[1]+  Terminated          sleep 60
```

To kill multiple instances of the same process, the killall command can be used:

```
$ sleep 60 &
[1] 1246
$ sleep 70 &
[2] 1247
$ killall sleep
[1]-  Terminated          sleep 60
[2]+  Terminated          sleep 70
```

Both pkill and killall work much in the same way as kill in that they send a terminating signal to the specified process(es). If no signal is provided, the default of SIGTERM is sent. However, kill only takes either a job or a process ID as an argument.

Signals can be specified either by:

Name:

```
$ kill -SIGHUP 1247
```

Number:

```
$ kill -1 1247
```

Option:

```
$ kill -s SIGHUP 1247
```

To have kill work in a similar way to pkill or killall (and save ourselves the commands to find out PIDs first) we can use command substitution:

```
$ kill -1 $(pgrep sleep)
```

As you should already know, an alternative syntax is kill -1 `pgrep sleep` .

Tip

For an exhaustive list of all kill signals and their codes, type kill -l into the terminal. Use -KILL (-9 or -s KILL) to kill rebel processes when any other signal fails.

Features of Terminal Multiplexers

Introduction

The tools and utilities seen in the previous lesson are very useful for process monitoring at large. However, a system administrator may need to go one step further. In this lesson we will discuss the concept of terminal multiplexers and learn about GNU Screen and tmux as — despite today's modern and great terminal emulators — multiplexers still preserve some cool, powerful features for a productive system administrator.

Features of Terminal Multiplexers

In electronics, a multiplexer (or mux) is a device that allows for multiple inputs to be connected to a single output. Thus, a terminal multiplexer gives us the ability to switch between different inputs as required. Although not exactly the same, screen and tmux share a series of common features:

Any successful invocation will result in at least a session which — in turn — will include at least a window. Windows contain programs.

Windows can be split into regions or panes — which can aid in productivity when working with various programs simultaneously.

Ease of control: to run most commands, they use a key combination — the so-called command prefix or command key — followed by another character.

Sessions can be detached from their current terminal (that is, programs are sent to the background and continue to run). This guarantees complete execution of programs no matter if we accidentally close a terminal, experience an occasional terminal freeze or even remote connection loss.

Socket connection.

Copy mode.

They are highly customizable.

GNU Screen

In the earlier days of Unix (1970s - 80s) computers basically consisted of terminals connected to a central computer. That was all, no multiple windows or tabs. And that was the reason behind the creation of GNU Screen in 1987: emulate multiple independent VT100 screens on a single physical terminal.

Windows

GNU Screen is invoked just by typing screen into the terminal. You will first see a welcome message:

GNU Screen version 4.05.00 (GNU) 10-Dec-16

Copyright (c) 2010 Juergen Weigert, Sadrul Habib Chowdhury
Copyright (c) 2008, 2009 Juergen Weigert, Michael Schroeder, Micah Cowan, Sadrul Habib Chowdhury
Copyright (c) 1993-2002, 2003, 2005, 2006, 2007 Juergen Weigert, Michael Schroeder
Copyright (c) 1987 Oliver Laumann
(...)

Press Space or Enter to close the message and you will be shown a command prompt:

\$

It may seem nothing has happened but the fact is that screen has already created and managing its first session and window. Screen's command prefix is Ctrl+a. To see all windows at the bottom of the terminal display, type Ctrl+a-w:

0*\$ bash

There it is, our one and only window so far! Note that the count starts at 0, though. To create another window, type Ctrl+a-c. You will see a new prompt. Let us start ps in that new window:

\$ ps

```
PID TTY      TIME CMD
974 pts/2    00:00:00 bash
981 pts/2    00:00:00 ps
```

and type Ctrl+a-w again:

0-\$ bash 1*\$ bash

There we have our two windows (note the asterisk indicating the one that is being shown at the moment). However, as they were started with Bash, they are both given the same name. Since we invoked ps in our current window, let us rename it with that same name. For that, you have to type Ctrl+a-A and write the new window name (ps) when prompted:

Set window's title to: ps

Now, let us create another window but provide it a name from the start: yetanotherwindow. This is done by invoking screen with the -t switch:

\$ screen -t yetanotherwindow

You can move between windows in different ways:

By using Ctrl+a-n (go to next window) and Ctrl+a-p (go to previous window).

By using Ctrl+a-number (go to window number number).

By using Ctrl+a-" to see a list of all windows. You can move up and down with the arrow keys and select the one you want with Enter:

Num	Name	Flags
0	bash	\$
1	ps	\$
2	yetanotherwindow	

While working with windows it is important to remember the following:

Windows run their programs completely independent of each other.

Programs will continue to run even if their window is not visible (also when the screen session is detached as we will see shortly).

To remove a window, simply terminate the program running in it (once the last window is removed, screen will itself terminate). Alternatively, use Ctrl+a-k while in the window you want to remove; you will be asked for confirmation:

Really kill this window [y/n]

Window 0 (bash) killed.

Regions

screen can divide a terminal screen up into multiple regions in which to accommodate windows. These divisions can be either horizontal (Ctrl+a-S) or vertical (Ctrl+a-|).

The only thing the new region will show is just -- at the bottom meaning it is empty:

```
1 ps --
To move to the new region, type Ctrl+a-Tab. Now you can add a window by any of the methods that
we have already seen, for example: Ctrl+a-2. Now the -- should have turned into 2
yetanotherwindow:
```

```
$ ps $
  PID TTY          TIME CMD
 1020 pts/2    00:00:00 bash
 1033 pts/2    00:00:00 ps
$ screen -t yetanotherwindow
```

```
1 ps 2 yetanotherwindow
Important aspects to keep in mind when working with regions are:
```

You move between regions by typing Ctrl+a-Tab.

You can terminate all regions except the current one with Ctrl+a-Q.

You can terminate the current region with Ctrl+a-X.

Terminating a region does not terminate its associated window.

Sessions

So far we have played with a few windows and regions, but all belonging to the same and only session. It is time to start playing with sessions. To see a list of all sessions, type screen -list or screen -ls:

```
$ screen -list
There is a screen on:
 1037.pts-0.debian  (08/24/19 13:53:35)  (Attached)
1 Socket in /run/screen/S-carol.
That is our only session so far:
```

```
PID
1037
```

```
Name
```

pts-0.debian (indicating the terminal — in our case a pseudo terminal slave — and the hostname).

Status
Attached

Let us create a new session giving it a more descriptive name:

```
$ screen -S "second session"
```

The terminal display will be cleared and you will be given a new prompt. You can check for sessions once more:

```
$ screen -ls
```

There are screens on:

```
1090.second session (08/24/19 14:38:35) (Attached)
```

```
1037.pts-0.debian (08/24/19 13:53:36) (Attached)
```

2 Sockets in /run/screen/S-carol.

To kill a session, quit out of all its windows or just type the command `screen -S SESSION-PID -X quit` (you can also provide the session name instead). Let us get rid of our first session:

```
$ screen -S 1037 -X quit
```

You will be sent back to your terminal prompt outside of screen. But remember, our second session is still alive:

```
$ screen -ls
```

There is a screen on:

```
1090.second session (08/24/19 14:38:35) (Detached)
```

1 Socket in /run/screen/S-carol.

However, since we killed its parent session, it is given a new label: Detached.

Session Detachment

For a number of reasons you may want to detach a screen session from its terminal:

To let your computer at work do its business and connect remotely later from home.

To share a session with other users.

You detach a session with the key combination `Ctrl+a-d`. You will be taken back into your terminal:

```
[detached from 1090.second session]
```

```
$
```

To attach again to the session, you use the command `screen -r SESSION-PID`. Alternatively, you can use the `SESSION-NAME` as we saw above. If there is only one detached session, neither is compulsory:

```
$ screen -r
```

This command suffices to reattach to our second session:

```
$ screen -ls
```

There is a screen on:

```
1090.second session (08/24/19 14:38:35) (Attached)
```

1 Socket in /run/screen/S-carol.

Important options for session reattaching:

```
-d -r
```

Reattach a session and — if necessary — detach it first.

-d -R

Same as -d -r but screen will even create the session first if it does not exist.

-d -RR

Same as -d -R. However, use the first session if more than one is available.

-D -r

Reattach a session. If necessary, detach and logout remotely first.

-D -R

If a session is running, then reattach (detaching and logging out remotely first if necessary). If it was not running create it and notify the user.

-D -RR

Same as -D -R — only stronger.

-d -m

Start screen in detached mode. This creates a new session but does not attach to it. This is useful for system startup scripts.

-D -m

Same as -d -m, but does not fork a new process. The command exits if the session terminates.

Read the manual pages for screen to find out about other options.

Copy & Paste: Scrollback Mode

GNU Screen features a copy or scrollback mode. Once entered, you can move the cursor in the current window and through the contents of its history using the arrow keys. You can mark text and copy it across windows. The steps to follow are:

Enter copy/scrollback mode: Ctrl+a-[,

Move to the beginning of the piece of text you want to copy using the arrow keys.

Mark the beginning of the piece of text you want to copy: Space.

Move to the end of the piece of text you want to copy using the arrow keys.

Mark the end of the piece of text you want to copy: Space.

Go to the window of your choice and paste the piece of text: Ctrl+a-].

Modify process execution priorities

Introduction

Operating systems able to run more than one process at the same time are called multi-tasking or multi-processing systems. Whilst true simultaneity only happens when more than one processing unit is available, even single processor systems can mimic simultaneity by switching between

processes very quickly. This technique is also employed in systems with many equivalent CPUs, or symmetric multi-processor (SMP) systems, given that the number of potential concurrent processes greatly exceeds the number of available processor units.

In fact, only one process at a time can control the CPU. However, most process activities are system calls, that is, the running process transfers CPU control to an operating system's process so it performs the requested operation. System calls are in charge of all inter-device communication, like memory allocations, reading and writing on filesystems, printing text on the screen, user interaction, network transfers, etc. Transferring CPU control during a system call allows the operating system to decide whether to return CPU control to the previous process or to hand it to another process. As modern CPUs can execute instructions much faster than most external hardware can communicate with each other, a new controlling process can do a lot of CPU work while previously requested hardware responses are still unavailable. To ensure maximum CPU harnessing, multi-processing operating systems keep a dynamic queue of active processes waiting for a CPU time slot.

Although they allow to significantly improve CPU time utilization, relying solely on system calls to switch between processes is not enough to achieve satisfactory multi-tasking performance. A process that makes no system calls could control the CPU indefinitely. This is why modern operating systems are also preemptive, that is, a running process can be put back in the queue so a more important process can control the CPU, even if the running process has not made a system call.

The Linux Scheduler

Linux, as a preemptive multi-processing operating system, implements a scheduler that organizes the process queue. More precisely, the scheduler also decides which queued thread will be executed — a process can branch out many independent threads — but process and thread are interchangeable terms in this context. Every process has two predicates that intervene on its scheduling: the scheduling policy and the scheduling priority.

There are two main types of scheduling policies: real-time policies and normal policies. Processes under a real-time policy are scheduled by their priority values directly. If a more important process becomes ready to run, a less important running process is preempted and the higher priority process takes control of the CPU. A lower priority process will gain CPU control only if higher priority processes are idle or waiting for hardware response.

Any real-time process has higher priority than a normal process. As a general purpose operating system, Linux runs just a few real-time processes. Most processes, including system and user programs, run under normal scheduling policies. Normal processes usually have the same priority value, but normal policies can define execution priority rules using another process predicate: the nice value. To avoid confusion with the dynamic priorities derived from nice values, scheduling priorities are usually called static scheduling priorities.

The Linux scheduler can be configured in many different ways and even more intricate ways of establishing priorities exist, but these general concepts always apply. When inspecting and tuning process scheduling, it is important to keep in mind that only processes under normal scheduling policy will be affected.

Reading Priorities

Linux reserves static priorities ranging from 0 to 99 for real-time processes and normal processes are assigned to static priorities ranging from 100 to 139, meaning that there are 39 different priority levels for normal processes. Lower values mean higher priority. The static priority of an active process can be found in the sched file, located in its respective directory inside the /proc filesystem:

```
$ grep ^prio /proc/1/sched
prio          :    120
```

As shown in the example, the line beginning with prio gives the priority value of the process (the PID 1 process is the init or the systemd process, the first process the kernel starts during system initialization). The standard priority for normal processes is 120, so that it can be decreased to 100 or increased to 139. The priorities of all running process can be verified with the command `ps -Al` or `ps -el`:

```
$ ps -el
 F S  UID  PID  PPID  C PRI  NI ADDR SZ WCHAN  TTY      TIME CMD
 4 S   0    1    0  0 80   0 - 9292 -   ?    00:00:00 systemd
 4 S   0   19    1  0 80   0 - 8817 -   ?    00:00:00 systemd-journal
 4 S  104   61    1  0 80   0 - 64097 -   ?    00:00:00 rsyslogd
 4 S   0   63    1  0 80   0 - 7244 -   ?    00:00:00 cron
 1 S   0  126    1  0 80   0 - 4031 -   ?    00:00:00 dhclient
 4 S   0  154    1  0 80   0 - 3937 - pts/0    00:00:00 agetty
 4 S   0  155    1  0 80   0 - 3937 - pts/1    00:00:00 agetty
 4 S   0  156    1  0 80   0 - 3937 - pts/2    00:00:00 agetty
 4 S   0  157    1  0 80   0 - 3937 - pts/3    00:00:00 agetty
 4 S   0  158    1  0 80   0 - 3937 - console 00:00:00 agetty
 4 S   0  160    1  0 80   0 - 16377 -   ?    00:00:00 sshd
 4 S   0  280    0  0 80   0 - 5301 -   ?    00:00:00 bash
 0 R   0  392  280  0 80   0 - 7221 -   ?    00:00:00 ps
```

The PRI column indicates the static priority assigned by the kernel. Note, however, that the priority value displayed by `ps` differs from that obtained in the previous example. Due to historical reasons, priorities displayed by `ps` range from -40 to 99 by default, so the actual priority is obtained by adding 40 to it (in particular, $80 + 40 = 120$).

It is also possible to continuously monitor processes currently being managed by the Linux kernel with program `top`. As with `ps`, `top` also displays the priority value differently. To make it easier to identify real-time processes, `top` subtracts the priority value by 100, thus making all real-time priorities negative, with a negative number or `rt` identifying them. Therefore, normal priorities displayed by `top` range from 0 to 39.

Note

To get more details from the `ps` command, additional options can be used. Compare the output from this command to the one from our previous example:

```
$ ps -e -o user,uid,comm,TTY,pid,ppid,pri,pmem,pcpu --sort=-pcpu | head
```

Process Niceness

Every normal process begins with a default nice value of 0 (priority 120). The nice name comes from the idea that “nicer” processes allow other processes to run before them in a particular execution queue. Nice numbers range from -20 (less nice, high priority) to 19 (more nice, low priority). Linux also allows the ability to assign different nice values to threads within the same process. The NI column in `ps` output indicates the nice number.

Only the root user can decrease the niceness of a process below zero. It's possible to start a process with a non-standard priority with the command `nice`. By default, `nice` changes the niceness to 10, but it can be specified with option `-n`:

```
$ nice -n 15 tar czf home_backup.tar.gz /home
```

In this example, the command `tar` is executed with a niceness of 15. The command `renice` can be

used to change the priority of a running process. The option -p indicates the PID number of the target process. For example:

```
# renice -10 -p 2164
```

2164 (process ID) old priority 0, new priority -10

The options -g and -u are used to modify all the processes of a specific group or user, respectively. With renice +5 -g users, the niceness of processes owned by users of the group users will be raised in five.

Besides renice, the priority of processes can be modified with other programs, like the program top. On the top main screen, the niceness of a process can be modified by pressing r and then the PID number of the process:

```
top - 11:55:21 up 23:38, 1 user, load average: 0,10, 0,04, 0,05
Tasks: 20 total, 1 running, 19 sleeping, 0 stopped, 0 zombie
%Cpu(s): 0,5 us, 0,3 sy, 0,0 ni, 99,0 id, 0,0 wa, 0,2 hi, 0,0 si, 0,0 st
KiB Mem : 4035808 total, 774700 free, 1612600 used, 1648508 buff/cache
KiB Swap: 7999828 total, 7738780 free, 261048 used. 2006688 avail Mem
PID to renice [default pid = 1]
  PID USER  PR  NI  VIRT  RES  SHR S %CPU %MEM  TIME+ COMMAND
    1 root    20   0 74232  7904 6416 S 0,000 0,196  0:00.12 systemd
   15 root    20   0 67436  6144 5568 S 0,000 0,152  0:00.03 systemd-journal
   21 root    20   0 61552  5628 5000 S 0,000 0,139  0:00.01 systemd-logind
   22 message+ 20   0 43540  4072 3620 S 0,000 0,101  0:00.03 dbus-daemon
   23 root    20   0 45652  6204 4992 S 0,000 0,154  0:00.06 wickedd-dhcp4
   24 root    20   0 45648  6276 5068 S 0,000 0,156  0:00.06 wickedd-auto4
   25 root    20   0 45648  6272 5060 S 0,000 0,155  0:00.06 wickedd-dhcp6
```

The message PID to renice [default pid = 1] appears with the first listed process selected by default. To change the priority of another process, type its PID and press Enter. Then, the message Renice PID 1 to value will appear (with the requested PID number) and a new nice value can be assigned.

Search text files using regular expressions

Introduction

String-searching algorithms are widely used by several data-processing tasks, so much that Unix-like operating systems have their own ubiquitous implementation: Regular expressions, often abbreviated to REs. Regular expressions consist of character sequences that make up a generic pattern used to locate and sometimes modify a corresponding sequence in a larger string of characters. Regular expressions greatly expand the ability to:

Write parsing rules to requests in HTTP servers, nginx in particular.

Write scripts that convert text-based datasets to another format.

Search for occurrences of interest in journal entries or documents.

Filter markup documents, keeping semantic content.

The simplest regular expression contains at least one atom. An atom, so named because it's the basic element of a regular expression, is just a character that may or may not have special meaning. Most ordinary characters are unambiguous, they retain their literal meaning, while others have special meaning:

. (dot)

Atom matches with any character.

^ (caret)

Atom matches with the beginning of a line.

\$ (dollar sign)

Atom matches with the end of a line.

For example, the regular expression `bc`, composed by the literal atoms `b` and `c`, can be found in the string `abcd`, but can not be found in the string `a1cd`. On the other hand, the regular expression `.c` can be found in both strings `abcd` and `a1cd`, as the dot `.` matches with any character.

The caret and dollar sign atoms are used when only matches at the beginning or at the end of the string are of interest. For that reason they are also called anchors. For example, `cd` can be found in `abcd`, but `^cd` can not. Similarly, `ab` can be found in `abcd`, but `ab$` can not. The caret `^` is a literal character except when at the beginning and `$` is a literal character except when at the end of the regular expression.

Bracket Expression

There is another type of atom named bracket expression. Although not a single character, brackets `[]` (including their content) are considered a single atom. A bracket expression usually is just a list of literal characters enclosed by `[]`, making the atom match any single character from the list. For example, the expression `[1b]` can be found in both strings `abcd` and `a1cd`. To specify characters the atom should not correspond to, the list must begin with `^`, as in `[^1b]`. It is also possible to specify ranges of characters in bracket expressions. For example, `[0–9]` matches digits 0 to 9 and `[a–z]` matches any lowercase letter. Ranges must be used with caution, as they might not be consistent across distinct locales.

Bracket expression lists also accept classes instead of just single characters and ranges. Traditional character classes are:

`[:alnum:]`

Represents an alphanumeric character.

`[:alpha:]`

Represents an alphabetic character.

`[:ascii:]`

Represents a character that fits into the ASCII character set.

`[:blank:]`

Represents a blank character, that is, a space or a tab.

`[:cntrl:]`

Represents a control character.

`[:digit:]`

Represents a digit (0 through 9).

`[:graph:]`

Represents any printable character except space.

`[lower:]`

Represents a lowercase character.

`[print:]`

Represents any printable character including space.

`[punct:]`

Represents any printable character which is not a space or an alphanumeric character.

`[space:]`

Represents white-space characters: space, form-feed (`\f`), newline (`\n`), carriage return (`\r`), horizontal tab (`\t`), and vertical tab (`\v`).

`[upper:]`

Represents an uppercase letter.

`[xdigit:]`

Represents hexadecimal digits (0 through F).

Character classes can be combined with single characters and ranges, but may not be used as an endpoint of a range. Also, character classes may be used only in bracket expressions, not as an independent atom outside the brackets.

Quantifiers

The reach of an atom, either a single character atom or a bracket atom, can be adjusted using an atom quantifier. Atom quantifiers define atom sequences, that is, matches occur when a contiguous repetition for the atom is found in the string. The substring corresponding to the match is called a piece. Notwithstanding, quantifiers and other features of regular expressions are treated differently depending on which standard is being used.

As defined by POSIX, there are two forms of regular expressions: “basic” regular expressions and “extended” regular expressions. Most text related programs in any conventional Linux distribution support both forms, so it is important to know their differences in order to avoid compatibility issues and to pick the most suitable implementation for the intended task.

The `*` quantifier has the same function in both basic and extended REs (atom occurs zero or more times) and it's a literal character if it appears at the beginning of the regular expression or if it's preceded by a backslash `\`. The plus sign quantifier `+` will select pieces containing one or more atom matches in sequence. With the question mark quantifier `?`, a match will occur if the corresponding atom appears once or if it doesn't appear at all. If preceded by a backslash `\`, their special meaning is not considered. Basic regular expressions also support `+` and `?` quantifiers, but they need to be preceded by a backslash. Unlike extended regular expressions, `+` and `?` by themselves are literal characters in basic regular expressions.

Bounds

A bound is an atom quantifier that, as the name implies, allows a user to specify precise quantity boundaries for an atom. In extended regular expressions, a bound may appear in three forms:

`{i}`

The atom must appear exactly *i* times (*i* an integer number). For example, `[[:blank:]]{2}` matches with exactly two blank characters.

`{i,}`

The atom must appear at least *i* times (*i* an integer number). For example, `[[:blank:]]{2,}` matches with any sequence of two or more blank characters.

`{i,j}`

The atom must appear at least *i* times and at most *j* times (*i* and *j* integer numbers, *j* greater than *i*). For example, `xyz{2,4}` matches the `xy` string followed by two to four of the `z` character.

In any case, if a substring matches with a regular expression and a longer substring starting at the same point also matches, the longer substring will be considered.

Basic regular expressions also support bounds, but the delimiters must be preceded by `\`: `\{` and `\}`. By themselves, `{` and `}` are interpreted as literal characters. A `\{` followed by a character other than a digit is a literal character, not the beginning of a bound.

Branches and Back References

Basic regular expressions also differ from extended regular expressions in another important aspect: an extended regular expression can be divided into branches, each one an independent regular expression. Branches are separated by `|` and the combined regular expression will match anything that corresponds to any of the branches. For example, `he|him` will match if either substring `he` or `him` are found in the string being examined. Basic regular expressions interpret `|` as a literal character. However, most programs supporting basic regular expressions will allow branches with `\|`.

An extended regular expression enclosed in `()` can be used in a back reference. For example, `([[:digit:]]+)\1` will match any regular expression that repeats itself at least once, because the `\1` in the expression is the back reference to the piece matched by the first parenthesized subexpression. If more than one parenthesized subexpression exist in the regular expression, they can be referenced with `\2`, `\3` and so on.

For basic REs, subexpressions must be enclosed by `\(` and `\)`, with `(` and `)` by themselves ordinary characters. The back reference indicator is used like in extended regular expressions.

Searching with Regular Expressions

The immediate benefit offered by regular expressions is to improve searches on filesystems and in text documents. The `-regex` option of command `find` allows to test every path in a directory hierarchy against a regular expression. For example,

```
$ find $HOME -regex '.*\/.*' -size +100M
```

searches for files greater than 100 megabytes (100 units of 1048576 bytes), but only in paths inside the user's home directory that do contain a match with `.*\/.*`, that is, a `/` surrounded by any other number of characters. In other words, only hidden files or files inside hidden directories will be listed, regardless of the position of `/` in the corresponding path. For case insensitive regular expressions, the `-iregex` option should be used instead:

```
$ find /usr/share/fonts -regextype posix-extended -iregex '.*(dejavu|liberation).*sans.*(italic|oblique).*
```

```
/usr/share/fonts/dejavu/DejaVuSansCondensed-BoldOblique.ttf
```

```
/usr/share/fonts/dejavu/DejaVuSansCondensed-Oblique.ttf
```

```
/usr/share/fonts/dejavu/DejaVuSans-BoldOblique.ttf
```

```
/usr/share/fonts/dejavu/DejaVuSans-Oblique.ttf
```

```
/usr/share/fonts/dejavu/DejaVuSansMono-BoldOblique.ttf
```

```
/usr/share/fonts/dejavu/DejaVuSansMono-Oblique.ttf
```

```
/usr/share/fonts/liberation/LiberationSans-BoldItalic.ttf
```

```
/usr/share/fonts/liberation/LiberationSans-Italic.ttf
```

In this example, the regular expression contains branches (written in extended style) to list only specific font files under the `/usr/share/fonts` directory hierarchy. Extended regular expressions are not supported by default, but `find` allows for them to be enabled with `-regextype posix-extended` or `-regextype egrep`. The default RE standard for `find` is `findutils-default`, which is virtually a basic regular expression clone.

It is often necessary to pass the output of a program to `command less` when it doesn't fit on the screen. `Command less` splits its input in pages, one screenful at a time, allowing the user to easily navigate the text up and down. In addition, `less` also allows a user to perform regular expression based searches. This feature is notably important because `less` is the default paginator used for many everyday tasks, like inspecting journal entries or consulting manual pages. When reading a manual page, for instance, pressing the `/` key will open a search prompt. This is a typical scenario in which regular expressions are useful, as command options are listed just after a page margin in the general manual page layout. However, the same option might appear many times through the text, making literal searches unfeasible. Regardless of that, typing `^[[:blank:]]*-o —` or more simply: `^ *-o —` in the search prompt will jump immediately to option the `-o` section (if it exists) after pressing Enter, thus allowing one to consult an option description more rapidly.

The Pattern Finder: grep

One of the most common uses of `grep` is to facilitate the inspection of long files, using the regular expression as a filter applied to each line. It can be used to show only the lines starting with a certain term. For example, `grep` can be used to investigate a configuration file for kernel modules, listing only option lines:

```
$ grep '^options' /etc/modprobe.d/alsa-base.conf
```

```
options snd-pcsp index=-2
```

```
options snd-usb-audio index=-2
```

```
options bt87x index=-2
```

```
options cx88_alsa index=-2
```

```
options snd-atiixp-modem index=-2
```

```
options snd-intel8x0m index=-2
```

```
options snd-via82xx-modem index=-2
```

The pipe `|` character can be employed to redirect the output of a command directly to `grep`'s input. The following example uses a bracket expression to select lines from `fdisk -l` output, starting with `Disk /dev/sda` or `Disk /dev/sdb`:

```
# fdisk -l | grep '^Disk /dev/sd[ab]'
```

```
Disk /dev/sda: 320.1 GB, 320072933376 bytes, 625142448 sectors
```

```
Disk /dev/sdb: 7998 MB, 7998537728 bytes, 15622144 sectors
```

The mere selection of lines with matches may not be appropriate for a particular task, requiring adjustments to `grep`'s behavior through its options. For example, option `-c` or `--count` tells `grep` to show how many lines had matches:

```
# fdisk -l | grep '^Disk /dev/sd[ab]' -c
```

The option can be placed before or after the regular expression. Other important grep options are:

-c or --count

Instead of displaying the search results, only display the total count for how many times a match occurs in any given file.

-i or --ignore-case

Turn the search case-insensitive.

-f FILE or --file=FILE

Indicate a file containing the regular expression to use.

-n or --line-number

Show the number of the line.

-v or --invert-match

Select every line, except those containing matches.

-H or --with-filename

Print also the name of the file containing the line.

-z or --null-data

Rather than have grep treat input and output data streams as separate lines (using the newline by default) instead take the input or output as a sequence of lines. When combining output from the find command using its `-print0` option with the grep command, the `-z` or `--null-data` option should be used to process the stream in the same manner.

Although activated by default when multiple file paths are given as input, the option `-H` is not activated for single files. That may be critical in special situations, like when grep is called directly by find, for instance:

```
$ find /usr/share/doc -type f -exec grep -i '3d modeling' "{}" \; | cut -c -100
```

artistic aspects of 3D modeling. Thus this might be the application you are

This major approach of 3D modeling has not been supported

oce is a C++ 3D modeling library. It can be used to develop CAD/CAM softwares, for instance
[FreeCad

In this example, find lists every file under /usr/share/doc then passes each one to grep, which in turn performs a case-insensitive search for 3d modeling inside the file. The pipe to cut is there just to limit output length to 100 columns. Note, however, that there is no way of knowing from which file the lines came from. This issue is solved by adding `-H` to grep:

```
$ find /usr/share/doc -type f -exec grep -i -H '3d modeling' "{}" \; | cut -c -100
```

/usr/share/doc/openscad/README.md:artistic aspects of 3D modeling. Thus this might be the applicatio

/usr/share/doc/opencsg/doc/publications.html:This major approach of 3D modeling has not been support

Now it is possible to identify the files where each match was found. To make the listing even more informative, leading and trailing lines can be added to lines with matches:

```
$ find /usr/share/doc -type f -exec grep -i -H -l '3d modeling' "{}" \; | cut -c -100
```

/usr/share/doc/openscad/README.md-application Blender), OpenSCAD focuses on the CAD aspects rather t

/usr/share/doc/openscad/README.md:artistic aspects of 3D modeling. Thus this might be the applicatio

/usr/share/doc/openscad/README.md-looking for when you are planning to create 3D models of machine p

/usr/share/doc/opencsg/doc/publications.html-3D graphics library for Constructive Solid Geometry (CS
/usr/share/doc/opencsg/doc/publications.html:This major approach of 3D modeling has not been support
/usr/share/doc/opencsg/doc/publications.html-by real-time computer graphics until recently.
The option -1 instructs grep to include one line before and one line after when it finds a line with a match. These extra lines are called context lines and are identified in the output by a minus sign after the file name. The same result can be obtained with -C 1 or --context=1 and other context line quantities may be indicated.

There are two complementary programs to grep: egrep and fgrep. The program egrep is equivalent to the command grep -E, which incorporates extra features other than the basic regular expressions. For example, with egrep it is possible to use extended regular expression features, like branching:

```
$ find /usr/share/doc -type f -exec egrep -i -H -1 '3d (modeling|printing)' "{}" \; | cut -c -100
/usr/share/doc/opencad/README.md-application Blender), OpenSCAD focuses on the CAD aspects rather t
/usr/share/doc/opencad/README.md:artistic aspects of 3D modeling. Thus this might be the applicatio
/usr/share/doc/opencad/README.md-looking for when you are planning to create 3D models of machine p
/usr/share/doc/opencad/RELEASE_NOTES.md-* Support for using 3D-Mouse / Joystick / Gamepad input dev
/usr/share/doc/opencad/RELEASE_NOTES.md:* 3D Printing support: Purchase from a print service partne
/usr/share/doc/opencad/RELEASE_NOTES.md-* New export file formats: SVG, 3MF, AMF
/usr/share/doc/opencsg/doc/publications.html-3D graphics library for Constructive Solid Geometry (CS
/usr/share/doc/opencsg/doc/publications.html:This major approach of 3D modeling has not been support
/usr/share/doc/opencsg/doc/publications.html-by real-time computer graphics until recently.
In this example either 3D modeling or 3D printing will match the expression, case-insensitive. To display only the parts of a text stream that match the expression used by egrep, use the -o option.
```

The program fgrep is equivalent to grep -F, that is, it does not parse regular expressions. It is useful in simple searches where the goal is to match a literal expression. Therefore, special characters like the dollar sign and the dot will be taken literally and not by their meanings in a regular expression.

The Stream Editor: sed

The purpose of the sed program is to modify text-based data in a non-interactive way. It means that all the editing is made by predefined instructions, not by arbitrarily typing directly into a text displayed on the screen. In modern terms, sed can be understood as a template parser: given a text as input, it places custom content at predefined positions or when it finds a match for a regular expression.

Sed, as the name implies, is well suited for text streamed through pipelines. Its basic syntax is sed -f SCRIPT when editing instructions are stored in the file SCRIPT or sed -e COMMANDS to execute COMMANDS directly from the command line. If neither -f or -e are present, sed uses the first non-option parameter as the script file. It is also possible to use a file as the input just by giving its path as an argument to sed.

sed instructions are composed of a single character, possibly preceded by an address or followed by one or more options, and are applied to each line at a time. Addresses can be a single line number, a regular expression, or a range of lines. For example, the first line of a text stream can be deleted

with 1d, where 1 specifies the line where the delete command d will be applied. To clarify sed 's usage, take the output of the command factor `seq 12`, which returns the prime factors for numbers 1 to 12:

```
$ factor `seq 12`
```

```
1:
2: 2
3: 3
4: 2 2
5: 5
6: 2 3
7: 7
8: 2 2 2
9: 3 3
10: 2 5
11: 11
12: 2 2 3
```

Deleting the first line with sed is accomplished by 1d:

```
$ factor `seq 12` | sed 1d
```

```
2: 2
3: 3
4: 2 2
5: 5
6: 2 3
7: 7
8: 2 2 2
9: 3 3
10: 2 5
11: 11
12: 2 2 3
```

A range of lines can be specified with a separating comma:

```
$ factor `seq 12` | sed 1,7d
```

```
8: 2 2 2
9: 3 3
10: 2 5
11: 11
12: 2 2 3
```

More than one instruction can be used in the same execution, separated by semicolons. In this case, however, it is important to enclose them with parenthesis so the semicolon is not interpreted by the shell:

```
$ factor `seq 12` | sed "1,7d;11d"
```

```
8: 2 2 2
9: 3 3
10: 2 5
12: 2 2 3
```

In this example, two deletion instructions were executed, first on lines ranging from 1 to 7 and then on line 11. An address can also be a regular expression, so only lines with a match will be affected by the instruction:

```
$ factor `seq 12` | sed "1d;/:.*2.*/d"
```

```
3: 3
5: 5
7: 7
9: 3 3
```

11: 11

The regular expression `:. *2.*` matches with any occurrence of the number 2 anywhere after a colon, causing the deletion of lines corresponding to numbers with 2 as a factor. With `sed`, anything placed between slashes (/) is considered a regular expression and by default all basic RE is supported. For example, `sed -e "/^#/d" /etc/services` shows the contents of the file `/etc/services` without the lines beginning with `#` (comment lines).

The delete instruction `d` is only one of the many editing instructions provided by `sed`. Instead of deleting a line, `sed` can replace it with a given text:

```
$ factor `seq 12` | sed "1d;/:.*2.*/c REMOVED"
```

REMOVED

3: 3

REMOVED

5: 5

REMOVED

7: 7

REMOVED

9: 3 3

REMOVED

11: 11

REMOVED

The instruction `c REMOVED` simply replaces a line with the text `REMOVED`. In the example's case, every line with a substring matching the regular expression `:. *2.*` is affected by instruction `c REMOVED`. Instruction `a TEXT` copies text indicated by `TEXT` to a new line after the line with a match. The instruction `r FILE` does the same, but copies the contents of the file indicated by `FILE`. Instruction `w` does the opposite of `r`, that is, the line will be appended to the indicated file.

By far the most used `sed` instruction is `s/FIND/REPLACE/`, which is used to replace a match to the regular expression `FIND` with text indicated by `REPLACE`. For example, the instruction `s/hda/sda/` replaces a substring matching the literal RE `hda` with `sda`. Only the first match found in the line will be replaced, unless the flag `g` is placed after the instruction, as in `s/hda/sda/g`.

A more realistic case study will help to illustrate `sed`'s features. Suppose a medical clinic wants to send text messages to its customers, reminding them of their scheduled appointments for the next day. A typical implementation scenario relies on a professional instant message service, which provides an API to access the system responsible for delivering the messages. These messages usually originate from the same system that runs the application controlling customer's appointments, triggered by a specific time of the day or some other event. In this hypothetical situation, the application could generate a file called `appointments.csv` containing tabulated data with all the appointments for the next day, then used by `sed` to render the text messages from a template file called `template.txt`. CSV files are a standard way of export data from database queries, so sample appointments could be given as follows:

```
$ cat appointments.csv
"NAME","TIME","PHONE"
"Carol","11am","55557777"
"Dave","2pm","33334444"
```

The first line holds the labels for each column, which will be used to match the tags inside the sample template file:

```
$ cat template.txt
Hey <NAME>, don't forget your appointment tomorrow at <TIME>.
```

The less than `<` and greater than `>` signs were put around labels just to help identify them as tags. The following Bash script parses all enqueued appointments using `template.txt` as the message template:


```
#!/bin/bash
```

```
TEMPLATE=`cat template.txt`
TAGS=(`sed -ne '1s/^"//;1s/","/\n/g;1s/"$//p' appointments.csv`)
mapfile -t -s 1 ROWS < appointments.csv
for (( r = 0; r < ${#ROWS[*]}; r++ ))
do
    MSG=$TEMPLATE
    VALS=(`sed -e 's/^"//;s/","/\n/g;s/"$//' <<<${ROWS[$r]}`)
    for (( c = 0; c < ${#TAGS[*]}; c++ ))
    do
        MSG=`sed -e "s/<${TAGS[$c]}>/${VALS[$c]}/g" <<<"$MSG"`
    done
    echo curl --data message=\"$MSG\" --data phone=\"$${VALS[2]}\" https://mysmsprovider/api
done
```

An actual production script would also handle authentication, error checking and logging, but the example has basic functionality to start with. The first instructions executed by sed are applied only to the first line — the address 1 in `1s/^"//;1s/","/\n/g;1s/"$//p` — to remove the leading and trailing quotes — `1s/^"//` and `1s/"$//` — and to replace field separators with a newline character: `1s/","/\n/g`. Only the first line is needed for loading column names, so non-matching lines will be suppressed by option `-n`, requiring flag `p` to be placed after the last sed command to print the matching line. The tags are then stored in the TAGS variable as a Bash array. Another Bash array variable is created by the command `mapfile` to store the lines containing the appointments in the array variable ROWS.

A for loop is employed to process each appointment line found in ROWS. Then, quotes and separators in the appointment — the appointment is in variable `${ROWS[$r]}` used as a here string — are replaced by sed, similarly to the commands used to load the tags. The separated values for the appointment are then stored in the array variable VALS, where array subscripts 0, 1 and 2 correspond to values for NAME, TIME and PHONE.

Finally, a nested for loop walks through the TAGS array and replaces each tag found in the template with its corresponding value in VALS. The MSG variable holds a copy of the rendered template, updated by the substitution command `s/<${TAGS[$c]}>/${VALS[$c]}/g` on every loop pass through TAGS.

This results in a rendered message like: "Hey Carol, don't forget your appointment tomorrow at 11am." The rendered message can then be sent as a parameter through a HTTP request with curl, as a mail message or any other similar method.

Combining grep and sed

Commands grep and sed can be used together when more complex text mining procedures are required. As a system administrator, you may want to inspect all the login attempts to a server, for example. The file `/var/log/wtmp` records all logins and logouts, whilst the file `/var/log/btmp` records the failed login attempts. They are written in a binary format, which can be read by the commands `last` and `lastb`, respectively.

The output of `lastb` shows not only the username used in the bad login attempt, but its IP address as well:

```
# lastb -d -a -n 10 --time-format notime
user ssh:notty      (00:00)  81.161.63.251
nrostagn ssh:notty  (00:00)  vmd60532.contaboserver.net
```

```

pi    ssh:notty    (00:00)  132.red-88-20-39.staticip.rima-tde.net
pi    ssh:notty    (00:00)  132.red-88-20-39.staticip.rima-tde.net
pi    ssh:notty    (00:00)  46.6.11.56
pi    ssh:notty    (00:00)  46.6.11.56

```

```

nps    ssh:notty    (00:00)  vmd60532.contaboserver.net
narmadan ssh:notty    (00:00)  vmd60532.contaboserver.net
nominati ssh:notty    (00:00)  vmd60532.contaboserver.net
nominati ssh:notty    (00:00)  vmd60532.contaboserver.net

```

Option -d translates the IP number to the corresponding hostname. The hostname may provide clues about the ISP or hosting service used to perform these bad login attempts. Option -a puts the hostname in the last column, which facilitates the filtering yet to be applied. Option --time-format notime suppresses the time when the login attempt occurred. Command lastb can take some time to finish if there were too many bad login attempts, so the output was limited to ten entries with the option -n 10.

Not all remote IPs have a hostname associated to it, so reverse DNS does not apply to them and they can be dismissed. Although you could write a regular expression to match the expected format for a hostname at the end of the line, it is probably simpler to write a regular expression to match with either a letter from the alphabet or with a single digit at the end of the line. The following example shows how the command grep takes the listing at its standard input and removes the lines without hostnames:

```

# lastb -d -a --time-format notime | grep -v '[0-9]$\ ' | head -n 10
nvidia  ssh:notty    (00:00)  vmd60532.contaboserver.net
n_tonson ssh:notty    (00:00)  vmd60532.contaboserver.net
nrostagn ssh:notty    (00:00)  vmd60532.contaboserver.net
pi      ssh:notty    (00:00)  132.red-88-20-39.staticip.rima-tde.net
pi      ssh:notty    (00:00)  132.red-88-20-39.staticip.rima-tde.net
nps     ssh:notty    (00:00)  vmd60532.contaboserver.net
narmadan ssh:notty    (00:00)  vmd60532.contaboserver.net
nominati ssh:notty    (00:00)  vmd60532.contaboserver.net
nominati ssh:notty    (00:00)  vmd60532.contaboserver.net
nominati ssh:notty    (00:00)  vmd60532.contaboserver.net

```

Command grep option -v shows only the lines that don't match with the given regular expression. A regular expression matching any line ending with a number (i.e. [0-9]\$) will capture only the entries without a hostname. Therefore, grep -v '[0-9]\$\ ' will show only the lines ending with a hostname.

The output can be filtered even further, by keeping only the domain name and removing the other parts from each line. Command sed can do it with a substitution command to replace the whole line with a back-reference to the domain name in it:

```

# lastb -d -a --time-format notime | grep -v '[0-9]$\ ' | sed -e 's/.* \(.*)$/\1/' | head -n 10
vmd60532.contaboserver.net
vmd60532.contaboserver.net
vmd60532.contaboserver.net
132.red-88-20-39.staticip.rima-tde.net
132.red-88-20-39.staticip.rima-tde.net
vmd60532.contaboserver.net
vmd60532.contaboserver.net
vmd60532.contaboserver.net
vmd60532.contaboserver.net
vmd60532.contaboserver.net

```

The escaped parenthesis in .* \(.*)\$ tells sed to remember that part of the line, that is, the part between the last space character and the end of the line. In the example, this part is referenced with \1 and used to replace the entire line.

It's clear that most remote hosts try to login more than once, thus the same domain name repeats

itself. To suppress the repeated entries, first they need to be sorted (with command `sort`) then passed to the command `uniq`:

```
# lastb -d -a --time-format notime | grep -v '[0-9]$\ ' | sed -e 's/.* \(.*\)$/\1/' | sort | uniq | head -n 10
116-25-254-113-on-nets.com
132.red-88-20-39.staticip.rima-tde.net
145-40-33-205.power-speed.at
tor.laquadrature.net
tor.momx.site
ua-83-226-233-154.bbcust.telenor.se
vmd38161.contaboserver.net
vmd60532.contaboserver.net
vmi488063.contaboserver.net
vmi515749.contaboserver.net
```

This shows how different commands can be combined to produce the desired outcome. The hostname list can then be used to write blocking firewall rules or to take other measures to enforce the security of the server.

Basic file editing

Introduction

In most Linux distributions, `vi` — abbreviation for “visual” — is pre-installed and it is the standard editor in the shell environment. `Vi` is an interactive text editor, it shows the file content on the screen as it is being edited. As such, it allows the user to move through and to make modifications anywhere in the document. However, unlike visual editors from the graphical desktop, the `vi` editor is a shell application with keyboard shortcuts to every editing task.

An alternative to `vi`, called `vim` (`vi` improved), is sometimes used as a modern replacement for `vi`. Among other improvements, `vim` offers support for syntax highlighting, multilevel undo/redo and multi-document editing. Although more resourceful, `vim` is fully backwards compatible with `vi`, making both indistinguishable for most tasks.

The standard way to start `vi` is to give it a path to a file as a parameter. To jump directly to a specific line, its number should be informed with a plus sign, as in `vi +9 /etc/fstab` to open `/etc/fstab/` and place the cursor at the 9th line. Without a number, the plus sign by itself places the cursor at the last line.

`vi`'s interface is very simple: all space available in the terminal window is occupied to present a file, normally informed as a command argument, to the user. The only visual clues are a footer line showing the current position of the cursor and a tilde `~` indicating where the file ends. There are different execution modes for `vi` where program behavior changes. The most common are: insert mode and normal mode.

Insert Mode

The insert mode is straightforward: text appears on the screen as it is typed on the keyboard. It is the type of interaction most users expect from a text editor, but it is not how `vi` first presents a document. To enter the insert mode, the user must execute an insertion command in the normal mode. The `Esc` key finishes the insert mode and returns to normal mode, the default `vi` mode.

Note

If you are interested to know more on the other execution modes, open vi and type:

:help vim-modes-intro

Normal Mode

Normal mode — also known as command mode — is how vi starts by default. In this mode, keyboard keys are associated with commands for navigation and text manipulation tasks. Most commands in this mode are unique keys. Some of the keys and their functions on normal mode are:

0, \$

Go to the beginning and end of the line.

1G, G

Go to the beginning and end of the document.

(,)

Go to the beginning and end of the sentence.

{, }

Go to the beginning and end of the paragraph.

w, W

Jump word and jump word including punctuation.

h, j, k, l

Left, down, up, right.

e or E

Go to the end of current word.

/, ?

Search forward and backwards.

i, I

Enter the insert mode before the current cursor position and at the beginning of the current line.

a, A

Enter the insert mode after the current cursor position and at the end of the current line.

o, O

Add a new line and enter the insert mode in the next line or in the previous line.

s, S

Erase the character under the cursor or the entire line and enter the insert mode.

c

Change the character(s) under the cursor.

r

Replace the character under the cursor.

x

Delete the selected characters or the character under the cursor.

`v, V`

Start a new selection with the current character or the entire line.

`y, yy`

Copy (yanks) the character(s) or the entire line.

`p, P`

Paste copied content, after or before the current position.

`u`

Undo the last action.

`Ctrl-R`

Redo the last action.

`ZZ`

Close and save.

`ZQ`

Close and do not save.

If preceded by a number, the command will be executed the same number of times. For example, press `3yy` to copy the current line plus the following two, press `d5w` to delete the current word and the following 4 words, and so on.

Most editing tasks are combinations of multiple commands. For example, the key sequence `vey` is used to copy a selection starting at the current position until the end of the current word. Command repetition can also be used in combinations, so `v3ey` would copy a selection starting at the current position until the end of the third word from there.

`vi` can organize copied text in registers, allowing to keep distinct contents at the same time. A register is specified by a character preceded by `"` and once created it's kept until the end of the current session. The key sequence `"ly` creates a register containing the current selection, which will be accessible through the key `l`. Then, the register `l` may be pasted with `"lp`.

There is also a way to set custom marks at arbitrary positions along the text, making it easier to quickly jump between them. Marks are created by pressing the key `m` and then a key to address the current position. With that done, the cursor will come back to the marked position when `'` followed by the chosen key are pressed.

Any key sequence can be recorded as a macro for future execution. A macro can be recorded, for example, to surround a selected text in double-quotes. First, a string of text is selected and the key `q` is pressed, followed by a register key to associate the macro with, like `d`. The line recording `@d` will appear in the footer line, indicating that the recording is on. It is assumed that some text is already selected, so the first command is `x` to remove (and automatically copy) the selected text. The key `i` is pressed to insert two double quotes at the current position, then `Esc` returns to normal mode. The last command is `P`, to re-insert the deleted selection just before the last double-quote. Pressing `q` again will end the recording. Now, a macro consisting of key sequence `x, i, "", Esc` and `P` will execute every time keys `@d` are pressed in normal mode, where `d` is the register key associated with the macro.

However, the macro will be available only during the current session. To make macros persistent, they should be stored in the configuration file. As most modern distributions use `vim` as the `vi` compatible editor, the user's configuration file is `~/.vimrc`. Inside `~/.vimrc`, the line `let @d = 'xi""P'` will set the register `d` to the key sequence inside single-quotes. The same register previously assigned to a macro can be used to paste its key sequence.

Colon Commands

The normal mode also supports another set of vi commands: the colon commands. Colon commands, as the name implies, are executed after pressing the colon key `:` in normal mode. Colon commands allow the user to perform searches, to save, to quit, to run shell commands, to change vi settings, etc. To go back to the normal mode, the command `:visual` must be executed or the Enter key pressed without any command. Some of the most common colon commands are indicated here (the initial `:` is not part of the command):

`:s/REGEX/TEXT/g`

Replaces all the occurrences of regular expression REGEX with TEXT in the current line. It accepts the same syntax of command `sed`, including addresses.

`:!`

Run a following shell command.

`:quit` or `:q`

Exit the program.

`:quit!` or `:q!`

Exit the program without saving.

`:wq`

Save and exit.

`:exit` or `:x` or `:e`

Save and exit, if needed.

`:visual`

Go back to navigation mode.

The standard vi program is capable of doing most text editing tasks, but any other non-graphical editor can be used to edit text files in the shell environment.

Tip

Novice users may have difficulty memorizing vi's command keys all at once. Distributions adopting vim also have the command `vimtutor`, which uses vim itself to open a step-by-step guide to the main activities. The file is an editable copy that can be used to practice the commands and progressively get used to them.

Alternative Editors

Users unfamiliar with vi may have difficulties adapting to it, since its operation is not intuitive. A simpler alternative is GNU nano, a small text editor that offers all basic text editing features like undo/redo, syntax coloring, interactive search-and-replace, auto-indentation, line numbers, word completion, file locking, backup files, and internationalization support. Unlike vi, all key presses are just inserted in the document being edited. Commands in nano are given by using the Ctrl key or the Meta key (depending on the system, Meta is Alt or ⌘).

Ctrl-6 or Meta-A

Start a new selection. It's also possible to create a selection by pressing Shift and moving the cursor.

Meta-6

Copy the current selection.

Ctrl-K

Cut the current selection.

Ctrl-U

Paste copied content.

Meta-U

Undo.

Meta-E

Redo.

Ctrl-\

Replace the text at the selection.

Ctrl-T

Start a spell-checking session for the document or current selection.

Emacs is another very popular text editor for the shell environment. Whilst text is inserted just by typing it, like in nano, navigation through the document is assisted by keyboard commands, like in vi. Emacs includes many features that makes it more than just a text editor. It is also an IDE (integrated development environment) capable of compiling, running, and testing programs. Emacs can be configured as an email, news or RSS client, making it an authentic productivity suite.

The shell itself will run a default text editor, usually vi, every time it is necessary. This is the case, for example, when `crontab -e` is executed to edit cronjobs. Bash uses the session variables `VISUAL` or `EDITOR` to find out the default text editor for the shell environment. For example, the command `export EDITOR=nano` defines nano as the default text editor in the current shell session. To make this change persistent across sessions, the command should be included in `~/.bash_profile`.

Create partitions and filesystems

Introduction

On any operating system, a disk needs to be partitioned before it can be used. A partition is a logical subset of the physical disk, and information about partitions are stored in a partition table. This table includes information about the first and last sectors of the partition and its type, and further details on each partition.

Usually each partition is seen by an operating system as a separate “disk”, even if they all reside in the same physical media. On Windows systems they are assigned letters like C: (historically the main disk), D: and so on. On Linux each partition is assigned to a directory under `/dev`, like `/dev/sda1` or `/dev/sda2`.

In this lesson, you will learn how to create, delete, restore and resize partitions using the three most common utilities (`fdisk`, `gdisk` and `parted`), how to create a filesystem on them and how to create and set up a swap partition or swap file to be used as virtual memory.

Understanding MBR and GPT

There are two main ways of storing partition information on hard disks. The first one is MBR (Master Boot Record), and the second one is GPT (GUID Partition Table).

MBR

This is a remnant from the early days of MS-DOS (more specifically, PC-DOS 2.0 from 1983) and for decades was the standard partitioning scheme on PCs. The partition table is stored on the first sector of a disk, called the Boot Sector, along with a boot loader, which on Linux systems is usually the GRUB bootloader. But MBR has a series of limitations that hinder its use on modern systems, like the inability to address disks of more than 2 TB in size, and the limit of only 4 primary partitions per disk.

GUID

A partitioning system that addresses many of the limitations of MBR. There is no practical limit on disk size, and the maximum number of partitions are limited only by the operating system itself. It is more commonly found on more modern machines that use UEFI instead of the old PC BIOS.

During system administration tasks it is highly possible that you will find both schemes in use, so it is important to know how to use the tools associated with each one to create, delete or modify partitions.

Managing MBR Partitions with FDISK

The standard utility for managing MBR partitions on Linux is fdisk. This is an interactive, menu-driven utility. To use it, type fdisk followed by the device name corresponding to the disk you want to edit. For example, the command

```
# fdisk /dev/sda
```

would edit the partition table of the first SATA-connected device (sda) on the system. Keep in mind that you need to specify the device corresponding to the physical disk, not one of its partitions (like /dev/sda1).

Note

All disk-related operations in this lesson need to be done as the user root (the system administrator), or with root privileges using sudo.

When invoked, fdisk will show a greeting, then a warning and it will wait for your commands.

```
# fdisk /dev/sda
```

Welcome to fdisk (util-linux 2.33.1).

Changes will remain in memory only, until you decide to write them.

Be careful before using the write command.

Command (m for help):

The warning is important. You can create, edit or delete partitions at will, but nothing will be written to disk unless you use the write (w) command. So you can “practice” without risk of losing data, as long as you stay clear of the w key. To exit fdisk without saving changes, use the q command.

Note

That being said, you should never practice on an important disk, as there are always risks. Use a spare external disk, or a USB flash drive instead.

Printing the Current Partition Table

The command `p` is used to print the current partition table. The output is something like this:

Command (m for help): `p`

Disk `/dev/sda`: 111.8 GiB, 120034123776 bytes, 234441648 sectors

Disk model: CT120BX500SSD1

Units: sectors of 1 * 512 = 512 bytes

Sector size (logical/physical): 512 bytes / 512 bytes

I/O size (minimum/optimal): 512 bytes / 512 bytes

Disklabel type: dos

Disk identifier: 0x97f8fef5

Device	Boot	Start	End	Sectors	Size	Id	Type
<code>/dev/sda1</code>		4096	226048942	226044847	107.8G	83	Linux
<code>/dev/sda2</code>		226048944	234437550	8388607	4G	82	Linux swap / Solaris

Here is the meaning of each column:

Device

The device assigned to the partition.

Boot

Shows whether the partition is “bootable” or not.

Start

The sector where the partition starts.

End

The sector where the partition ends.

Sectors

The total number of sectors in the partition. Multiply it by the sector size to get the partition size in bytes.

Size

The size of the partition in “human readable” format. In the example above, values are in gigabytes.

Id

The numerical value representing the partition type.

Type

The description for the partition type.

Primary vs Extended Partitions

On an MBR disk, you can have 2 main types of partitions, primary and extended. Like we said before, you can have only 4 primary partitions on the disk, and if you want to make the disk “bootable”, the first partition must be a primary one.

One way to work around this limitation is to create an extended partition that acts as a container for logical partitions. You could have, for example, a primary partition, an extended partition occupying the remainder of the disk space and five logical partitions inside it.

For an operating system like Linux, primary and extended partitions are treated exactly in the same way, so there are no “advantages” of using one over the other.

Creating a Partition

To create a partition, use the `n` command. By default, partitions will be created at the start of unallocated space on the disk. You will be asked for the partition type (primary or extended), first sector and last sector.

For the first sector, you can usually accept the default value suggested by fdisk, unless you need a partition to start at a specific sector. Instead of specifying the last sector, you can specify a size followed by the letters K, M, G, T or P (Kilo, Mega, Giga, Tera or Peta). So, if you want to create a 1 GB partition, you could specify +1G as the Last sector, and fdisk will size the partition accordingly. See this example for the creation of a primary partition:

Command (m for help): n

Partition type

- p primary (0 primary, 0 extended, 4 free)
- e extended (container for logical partitions)

Select (default p): p

Partition number (1-4, default 1): 1

First sector (2048-3903577, default 2048): 2048

Last sector, +/-sectors or +/-size{K,M,G,T,P} (2048-3903577, default 3903577): +1G

Checking for Unallocated Space

If you do not know how much free space there is on the disk, you can use the F command to show the unallocated space, like so:

Command (m for help): F

Unpartitioned space /dev/sdd: 881 MiB, 923841536 bytes, 1804378 sectors

Units: sectors of 1 * 512 = 512 bytes

Sector size (logical/physical): 512 bytes / 512 bytes

Start	End	Sectors	Size
2099200	3903577	1804378	881M

Deleting Partitions

To delete a partition, use the d command. fdisk will ask you for the number of the partition you want to delete, unless there is only one partition on the disk. In this case, this partition will be selected and deleted immediately.

Be aware that if you delete an extended partition, all the logical partitions inside it will also be deleted.

Mind the Gap!

Keep in mind that when creating a new partition with fdisk, the maximum size will be limited to the maximum amount of contiguous unallocated space on the disk. Say, for example, that you have the following partition map:

Device	Boot	Start	End	Sectors	Size	Id	Type
/dev/sdd1		2048	1050623	1048576	512M	83	Linux
/dev/sdd2		1050624	2099199	1048576	512M	83	Linux
/dev/sdd3		2099200	3147775	1048576	512M	83	Linux

Then you delete partition 2 and check for free space:

Command (m for help): F

Unpartitioned space /dev/sdd: 881 MiB, 923841536 bytes, 1804378 sectors

Units: sectors of 1 * 512 = 512 bytes

Sector size (logical/physical): 512 bytes / 512 bytes

Start	End	Sectors	Size
1050624	2099199	1048576	512M
3147776	3903577	755802	369M

Adding up the size of the unallocated space, in theory we have 881 MB available. But see what happens when we try to create a 700 MB partition:

Command (m for help): n

Partition type

- p primary (2 primary, 0 extended, 2 free)
- e extended (container for logical partitions)

Select (default p): p

Partition number (2,4, default 2): 2

First sector (1050624-3903577, default 1050624):

Last sector, +/-sectors or +/-size{K,M,G,T,P} (1050624-2099199, default 2099199): +700M

Value out of range.

That happens because the largest contiguous unallocated space on the disk is the 512 MB block that belonged to partition 2. Your new partition cannot “reach over” partition 3 to use some of the unallocated space after it.

Changing the Partition Type

Occasionally, you may need to change the partition type, especially when dealing with disks that will be used on other operating systems and platforms. This is done with the command `t`, followed by the number of the partition you wish to change.

The partition type must be specified by its corresponding hexadecimal code, and you can see a list of all the valid codes by using the command `l`.

Do not confuse the partition type with the filesystem used on it. Although at first there was a relation between them, today you cannot assume this to be true. A Linux partition, for example, can contain any Linux-native filesystem, like `ext4` or `ReiserFS`.

Tip

Linux partitions are type 83 (Linux). Swap partitions are type 82 (Linux Swap).

Managing GUID Partitions with GDISK

The utility `gdisk` is the equivalent of `fdisk` when dealing with GPT partitioned disks. In fact, the interface is modeled after `fdisk`, with an interactive prompt and the same (or very similar) commands.

Printing the Current Partition Table

The command `p` is used to print the current partition table. The output is something like this:

Command (? for help): p

Disk /dev/sdb: 3903578 sectors, 1.9 GiB

Model: DataTraveler 2.0

Sector size (logical/physical): 512/512 bytes

Disk identifier (GUID): AB41B5AA-A217-4D1E-8200-E062C54285BE

Partition table holds up to 128 entries

Main partition table begins at sector 2 and ends at sector 33

First usable sector is 34, last usable sector is 3903544

Partitions will be aligned on 2048-sector boundaries

Total free space is 1282071 sectors (626.0 MiB)

Number	Start (sector)	End (sector)	Size	Code	Name
1	2048	2099199	1024.0 MiB	8300	Linux filesystem
2	2623488	3147775	256.0 MiB	8300	Linux filesystem

Right from the start, we notice a few different things:

Each disk has a unique Disk Identifier (GUID). This is a 128 bit hexadecimal number, assigned

randomly when the partition table is created. Since there are 3.4×10^{38} possible values to this number, the chances that 2 random disks have the same GUID are pretty slim. The GUID can be used to identify which filesystems to mount at boot time (and where), eliminating the need to use the device path to do so (like `/dev/sdb`).

See the phrase Partition table holds up to 128 entries? That's right, you can have up to 128 partitions on a GPT disk. Because of this, there is no need for primary and extended partitions.

The free space is listed on the last line, so there is no need for an equivalent of the `F` command from `fdisk`.

Creating a Partition

The command to create a partition is `n`, just as in `fdisk`. The main difference is that besides the partition number and the first and last sector (or size), you can also specify the partition type during the creation. GPT partitions support many more types than MBR. You can check a list of all the supported types by using the `l` command.

Deleting a Partition

To delete a partition, type `d` and the partition number. Unlike `fdisk`, the first partition will not be automatically selected if it is the only one on the disk.

On GPT disks, partitions can be easily reordered, or "sorted", to avoid gaps in the numbering sequence. To do this, simply use the `s` command. For example, imagine a disk with the following partition table:

Number	Start (sector)	End (sector)	Size	Code	Name
1	2048	2099199	1024.0 MiB	8300	Linux filesystem
2	2099200	2361343	128.0 MiB	8300	Linux filesystem
3	2361344	2623487	128.0 MiB	8300	Linux filesystem

If you delete the second partition, the table would become:

Number	Start (sector)	End (sector)	Size	Code	Name
1	2048	2099199	1024.0 MiB	8300	Linux filesystem
3	2361344	2623487	128.0 MiB	8300	Linux filesystem

If you use the `s` command, it would become:

Number	Start (sector)	End (sector)	Size	Code	Name
1	2048	2099199	1024.0 MiB	8300	Linux filesystem
2	2361344	2623487	128.0 MiB	8300	Linux filesystem

Notice that the third partition became the second one.

Gap? What Gap?

Unlike MBR disks, when creating a partition on GPT disks the size is not limited by the maximum amount of contiguous unallocated space. You can use every last bit of a free sector, no matter where it is located on the disk.

Recovery Options

GPT disks store backup copies of the GPT header and partition table, making it easy to recover disks in case this data has been damaged. `gdisk` provides features to aid in those recovery tasks, accessed with the `r` command.

You can rebuild a corrupt main GPT header or partition table with `b` and `c`, respectively, or use the main header and table to rebuild a backup with `d` and `e`. You can also convert a MBR to a GPT with `f`, and do the opposite with `g`, among other operations. Type `?` in the recovery menu to get a list of all the available recovery commands and descriptions about what they do.

Creating File Systems

Partitioning the disk is only the first step towards being able to use a disk. After that, you will need to format the partition with a filesystem before using it to store data.

A filesystem controls how the data is stored and accessed on the disk. Linux supports many filesystems, some native, like the ext (Extended Filesystem) family, while others come from other operating systems like FAT from MS-DOS, NTFS from Windows NT, HFS and HFS+ from Mac OS, etc.

The standard tool used to create a filesystem on Linux is `mkfs`, which comes in many “flavors” according to the filesystem it needs to work with.

Creating an ext2/ext3/ext4 Filesystem

The Extended Filesystem (ext) was the first filesystem for Linux, and through the years was replaced by new versions called ext2, ext3 and ext4, which is currently the default filesystem for many Linux distributions.

The utilities `mkfs.ext2`, `mkfs.ext3` and `mkfs.ext4` are used to create ext2, ext3 and ext4 filesystems. In fact, all of these “utilities” exist only as symbolic links to another utility called `mke2fs`. `mke2fs` alters its defaults according to the name it is called by. As such, they all have the same behavior and command line parameters.

The most simple form of usage is:

```
# mkfs.ext2 TARGET
```

Where TARGET is the name of the partition where the filesystem should be created. For example, to create an ext3 filesystem on `/dev/sdb1` the command would be:

```
# mkfs.ext3 /dev/sdb1
```

Instead of using the command corresponding to the filesystem you wish to create, you can pass the `-t` parameter to `mke2fs` followed by the filesystem name. For example, the following commands are equivalent, and will create an ext4 filesystem on `/dev/sdb1`.

```
# mkfs.ext4 /dev/sdb1
```

```
# mke2fs -t ext4 /dev/sdb1
```

Command Line Parameters

`mke2fs` supports a wide range of command line parameters and options. Here are some of the most significant ones. All of them also apply to `mkfs.ext2`, `mkfs.ext3` and `mkfs.ext4`:

`-b SIZE`

Sets the size of the data blocks in the device to SIZE, which can be 1024, 2048 or 4096 bytes per block.

`-c`

Checks the target device for bad blocks before creating the filesystem. You can run a thorough, but much slower check by passing this parameter twice, as in `mkfs.ext4 -c -c TARGET`.

`-d DIRECTORY`

Copies the contents of the specified directory to the root of the new filesystem. Useful if you need to “pre-populate” the disk with a predefined set of files.

`-F`

Danger, Will Robinson! This option will force `mke2fs` to create a filesystem, even if the other options

passed to it or the target are dangerous or make no sense at all. If specified twice (as in -F -F) it can even be used to create a filesystem on a device which is mounted or in use, which is a very, very bad thing to do.

-L VOLUME_LABEL

Will set the volume label to the one specified in VOLUME_LABEL. This label must be at most 16 characters long.

-n

This is a truly useful option that simulates the creation of the filesystem, and displays what would be done if executed without the n option. Think of it as a “trial” mode. Good to check things out before actually committing any changes to disk.

-q

Quiet mode. mke2fs will run normally, but will not produce any output to the terminal. Useful when running mke2fs from a script.

-U ID

This will set the UUID (Universally Unique Identifier) of a partition to the value specified as ID. UUIDs are 128 bit numbers in hexadecimal notation that serve to uniquely identify a partition to the operating system. This number is specified as a 32-digit string in the format 8-4-4-4-12, meaning 8 digits, hyphen, 4 digits, hyphen, 4 digits, hyphen, 4 digits, hyphen, 12 digits, like D249E380-7719-45A1-813C-35186883987E. Instead of an ID you can also specify parameters like clear to clear the filesystem UUID, random, to use a randomly generated UUID, or time to create a time-based UUID.

-V

Verbose mode, prints much more information during operation than usual. Useful for debugging purposes.

Creating an XFS Filesystem

XFS is a high-performance filesystem originally developed by Silicon Graphics in 1993 for its IRIX operating system. Due to its performance and reliability features, it is commonly used for servers and other environments that require high (or guaranteed) filesystem bandwidth.

Tools for managing XFS filesystems are part of the xfsprogs package. This package may need to be installed manually, as it is not included by default in some Linux distributions. Others, like Red Hat Enterprise Linux 7, use XFS as the default filesystem.

XFS filesystems are divided into at least 2 parts, a log section where a log of all filesystem operations (commonly called a Journal) are maintained, and the data section. The log section may be located inside the data section (the default behavior), or even on a separate disk altogether, for better performance and reliability.

The most basic command to create an XFS filesystem is mkfs.xfs TARGET, where TARGET is the partition you want the filesystem to be created in. For example: mkfs.xfs /dev/sda1.

As in mke2fs, mkfs.xfs supports a number of command line options. Here are some of the most common ones.

-b size=VALUE

Sets the block size on the filesystem, in bytes, to the one specified in VALUE. The default value is

4096 bytes (4 KiB), the minimum is 512, and the maximum is 65536 (64 KiB).

-m crc=VALUE

Parameters starting with **-m** are metadata options. This one enables (if **VALUE** is 1) or disables (if **VALUE** is 0) the use of CRC32c checks to verify the integrity of all metadata on the disk. This enables better error detection and recovery from crashes related to hardware issues, so it is enabled by default. The performance impact of this check should be minimal, so normally there is no reason to disable it.

-m uuid=VALUE

Sets the partition UUID to the one specified as **VALUE**. Remember that UUIDs are 32-character (128 bits) numbers in hexadecimal base, specified in groups of 8, 4, 4, 4 and 12 digits separated by dashes, like 1E83E3A3-3AE9-4AAC-BF7E-29DFFECD36C0.

-f

Force the creation of a filesystem on the target device even if a filesystem is detected on it.

-l logdev=DEVICE

This will put the log section of the filesystem on the specified device, instead of inside the data section.

-l size=VALUE

This will set the size of the log section to the one specified in **VALUE**. The size can be specified in bytes, and suffixes like **m** or **g** can be used. **-l size=10m**, for example, will limit the log section to 10 Megabytes.

-q

Quiet mode. In this mode, **mkfs.xfs** will not print the parameters of the file system being created.

-L LABEL

Sets the filesystem label, which can be at most 12 characters long.

-N

Similar to the **-n** parameter of **mke2fs**, will make **mkfs.xfs** print all the parameters for the creation of the file system, without actually creating it.

Creating a FAT or VFAT Filesystem

exFAT is a filesystem created by Microsoft in 2006 that addresses one of the most important limitations of FAT32: file and disk size. On exFAT, the maximum file size is 16 exabytes (from 4 GB on FAT32), and the maximum disk size is 128 petabytes.

As it is well supported by all three major operating systems (Windows, Linux and mac OS), it is a good choice where interoperability is needed, like on large capacity flash drives, memory cards and external disks. In fact, it is the default filesystem, as defined by the SD Association, for SDXC memory cards larger than 32 GB.

The default utility for creating exFAT filesystems is **mkfs.exfat**, which is a link to **mkexfatfs**. The most basic command is **mkfs.exfat TARGET**, where **TARGET** is the partition you want the filesystem to be created in. For example: **mkfs.exfat /dev/sdb2**.

Contrary to the other utilities discussed in this lesson, **mkfs.exfat** has very few command line options. They are:

-i VOL_ID

Sets the Volume ID to the value specified in VOL_ID. This is a 32-Bit hexadecimal number. If not defined, an ID based on the current time is set.

-n NAME

Sets the volume label, or name. This can have up to 15 characters, and the default is no name.

-p SECTOR

Specifies the first sector of the first partition on the disk. This is an optional value, and the default is zero.

-s SECTORS

Defines the number of physical sectors per cluster of allocation. This must be a power of two, like 1, 2, 4, 8, and so on.

Getting to Know the Btrfs Filesystem

Btrfs (officially the B-Tree Filesystem, pronounced as “Butter FS”, “Better FS” or even “Butterfuss”, your pick) is a filesystem that has been in development since 2007 specifically for Linux by the Oracle Corporation and other companies, including Fujitsu, Red Hat, Intel and SUSE, among others.

There are many features that make Btrfs attractive on modern systems where massive amounts of storage are common. Among these are multiple device support (including striping, mirroring and striping+mirroring, as in a RAID setup), transparent compression, SSD optimizations, incremental backups, snapshots, online defragmentation, offline checks, support for subvolumes (with quotas), deduplication and much more.

As it is a copy-on-write filesystem it is very resilient to crashes. And on top of that, Btrfs is simple to use, and well supported by many Linux distributions. And some of them, like SUSE, use it as the default filesystem.

Note

On a traditional filesystem, when you want to overwrite part of a file the new data is put directly over the old data that it is replacing. On a copy-on-write filesystem the new data is written to free space on disk, then the file’s original metadata is updated to refer to the new data and only then the old data is freed up, as it is no longer needed. This reduces the chances of data loss in case of a crash, as the old data is only discarded after the filesystem is absolutely sure that it is no longer needed and the new data is in place.

Creating a Btrfs Filesystem

The utility mkfs.btrfs is used to create a Btrfs filesystem. Using the command without any options creates a Btrfs filesystem on a given device, like so:

```
# mkfs.btrfs /dev/sdb1
```

Tip

If you do not have the mkfs.btrfs utility on your system, look for btrfs-progs in your distribution’s package manager.

You can use the -L to set a label (or name) for your filesystem. Btrfs labels can have up to 256 characters, except for newlines:

```
# mkfs.btrfs /dev/sdb1 -L "New Disk"
```

Tip

Enclose the label in quotes (like above) if it contains spaces.

Note this peculiar thing about Btrfs: you can pass multiple devices to the mkfs.btrfs command.

Passing more than one device will span the filesystem over all the devices which is similar to a RAID or LVM setup. To specify how metadata will be distributed in the disk array, use the -m parameter. Valid parameters are raid0, raid1, raid5, raid6, raid10, single and dup.

For example, to create a filesystem spanning /dev/sdb1 and /dev/sdc1, concatenating the two partitions into one big partition, use:

```
# mkfs.btrfs -d single -m single /dev/sdb /dev/sdc
```

Warning

Filesystems spanning multiple partitions such as above might look advantageous at first but are not a good idea from a data safety standpoint, as a failure on a single disk of the array means certain data loss. The risk gets bigger the more disks you use, as you also have more possible points of failure.

Managing Subvolumes

Subvolumes are like filesystems inside filesystems. Think of them as a directory which can be mounted as (and treated like) a separate filesystem. Subvolumes make organization and system administration easier, as each one of them can have separate quotas or snapshot rules.

Note

Subvolumes are not partitions. A partition allocates a fixed space on a drive. This can lead to problems further down the line, like one partition running out of space when another one has plenty of space left. Not so with subvolumes, as they “share” the free space from their root filesystem, and grow as needed.

Suppose you have a Btrfs filesystem mounted on /mnt/disk, and you wish to create a subvolume inside it to store your backups. Let's call it BKP:

```
# btrfs subvolume create /mnt/disk/BKP
```

Next we list the contents of the /mnt/disk filesystem. You will see that we have a new directory, named after our subvolume.

```
$ ls -lh /mnt/disk/
total 0
drwxr-xr-x 1 root  root   0 jul 13 17:35 BKP
drwxrwxr-x 1 carol carol 988 jul 13 17:30 Images
```

Note

Yes, subvolumes can also be accessed like any other directory.

We can check that the subvolume is active, with the command:

```
# btrfs subvolume show /mnt/disk/BKP/
Name:          BKP
UUID:          e90a1afe-69fa-da4f-9764-3384f66fa32e
Parent UUID:    -
Received UUID:  -
Creation time:  2019-07-13 17:35:40 -0300
Subvolume ID:   260
Generation:     23
Gen at creation: 22
Parent ID:      5
Top level ID:   5
Flags:         -
Snapshot(s):
```

You can mount the subvolume on /mnt/BKP by passing the -t btrfs -o subvol=NAME parameter to the mount command:

```
# mount -t btrfs -o subvol=BKP /dev/sdb1 /mnt/bkp
```

Note

The -t parameter specifies the filesystem type to be mounted.

Working with Snapshots

Snapshots are just like subvolumes, but pre-populated with the contents from the volume from which the snapshot was taken.

When created, a snapshot and the original volume have exactly the same content. But from that point in time, they will diverge. Changes made to the original volume (like files added, renamed or deleted) will not be reflected on the snapshot, and vice-versa.

Keep in mind that a snapshot does not duplicate the files, and initially takes almost no disk space. It simply duplicates the filesystem tree, while pointing to the original data.

The command to create a snapshot is the same used to create a subvolume, just add the snapshot parameter after btrfs subvolume. The command below will create a snapshot of the Btrfs filesystem mounted in /mnt/disk in /mnt/disk/snap:

```
# btrfs subvolume snapshot /mnt/disk /mnt/disk/snap
```

Now, imagine you have the following contents in /mnt/disk:

```
$ ls -lh
total 2,8M
-rw-rw-r-- 1 carol carol 109K jul 10 16:22 Galaxy_Note_10.png
-rw-rw-r-- 1 carol carol 484K jul  5 15:01 geminoid2.jpg
-rw-rw-r-- 1 carol carol 429K jul  5 14:52 geminoid.jpg
-rw-rw-r-- 1 carol carol 467K jul  2 11:48 LG-G8S-ThinQ-Mirror-White.jpg
-rw-rw-r-- 1 carol carol 654K jul  2 11:39 LG-G8S-ThinQ-Range.jpg
-rw-rw-r-- 1 carol carol  94K jul  2 15:43 Memoji_Comparativo.jpg
-rw-rw-r-- 1 carol carol 112K jul 10 16:20 Note10Plus.jpg
drwx----- 1 carol carol  366 jul 13 17:56 snap
-rw-rw-r-- 1 carol carol 118K jul 11 16:36 Twitter_Down_20190711.jpg
-rw-rw-r-- 1 carol carol 324K jul  2 15:22 Xiaomi_Memoji.png
```

Notice the snap directory, containing the snapshot. Now let us remove some files, and check the directory contents:

```
$ rm LG-G8S-ThinQ-*
```

```
$ ls -lh
total 1,7M
-rw-rw-r-- 1 carol carol 109K jul 10 16:22 Galaxy_Note_10.png
-rw-rw-r-- 1 carol carol 484K jul  5 15:01 geminoid2.jpg
-rw-rw-r-- 1 carol carol 429K jul  5 14:52 geminoid.jpg
-rw-rw-r-- 1 carol carol  94K jul  2 15:43 Memoji_Comparativo.jpg
-rw-rw-r-- 1 carol carol 112K jul 10 16:20 Note10Plus.jpg
drwx----- 1 carol carol  366 jul 13 17:56 snap
-rw-rw-r-- 1 carol carol 118K jul 11 16:36 Twitter_Down_20190711.jpg
-rw-rw-r-- 1 carol carol 324K jul  2 15:22 Xiaomi_Memoji.png
```

However, if you check inside the snap directory, the files you deleted are still there and can be restored if needed.

```
$ ls -lh snap/
total 2,8M
-rw-rw-r-- 1 carol carol 109K jul 10 16:22 Galaxy_Note_10.png
-rw-rw-r-- 1 carol carol 484K jul  5 15:01 geminoid2.jpg
-rw-rw-r-- 1 carol carol 429K jul  5 14:52 geminoid.jpg
-rw-rw-r-- 1 carol carol 467K jul  2 11:48 LG-G8S-ThinQ-Mirror-White.jpg
```

```
-rw-rw-r-- 1 carol carol 654K jul  2 11:39 LG-G8S-ThinQ-Range.jpg
-rw-rw-r-- 1 carol carol  94K jul  2 15:43 Memoji_Comparativo.jpg
-rw-rw-r-- 1 carol carol 112K jul 10 16:20 Note10Plus.jpg
-rw-rw-r-- 1 carol carol 118K jul 11 16:36 Twitter_Down_20190711.jpg
-rw-rw-r-- 1 carol carol 324K jul  2 15:22 Xiaomi_Memoji.png
```

It is also possible to create read-only snapshots. They work exactly like writable snapshots, with the difference that the contents of the snapshot cannot be changed, they are “frozen” in time. Just add the `-r` parameter when creating the snapshot:

```
# btrfs subvolume snapshot -r /mnt/disk /mnt/disk/snap
```

A Few Words on Compression

Btrfs supports transparent file compression, with three different algorithms available to the user. This is done automatically on a per-file basis, as long as the filesystem is mounted with the `-o compress` option. The algorithms are smart enough to detect incompressible files and will not try to compress them, saving system resources. So on a single directory you may have compressed and uncompressed files together. The default compression algorithm is ZLIB, but LZO (faster, worse compression ratio) or ZSTD (faster than ZLIB, comparable compression) are available, with multiple compression levels (see the corresponding objective on mount options).

Managing Partitions with GNU Parted

GNU Parted is a very powerful partition editor (hence the name) that can be used to create, delete, move, resize, rescue and copy partitions. It can work with both GPT and MBR disks, and cover almost all of your disk management needs.

There are many graphical front-ends that make working with parted much easier, like GParted for GNOME-based desktop environments and the KDE Partition Manager for KDE Desktops. However, you should learn how to use parted on the command line, since in a server setting you can never count on a graphical desktop environment being available.

Warning

Unlike `fdisk` and `gdisk`, parted makes changes to the disk immediately after the command is issued, without waiting for another command to write the changes to disk. When practicing, it is wise to do so on an empty or spare disk or flash drive, so there is no risk of data loss should you make a mistake.

The simplest way to start using parted is by typing `parted DEVICE`, where `DEVICE` is the device you want to manage (`parted /dev/sdb`). The program starts an interactive command line interface like `fdisk` and `gdisk` with a `(parted)` prompt for you to enter commands.

```
# parted /dev/sdb
GNU Parted 3.2
Using /dev/sdb
Welcome to GNU Parted! Type 'help' to view a list of commands.
```

```
(parted)
```

Warning

Be careful! If you do not specify a device, parted will automatically select the primary disk (usually `/dev/sda`) to work with.

Selecting Disks

To switch to a different disk than the one specified on the command line, you can use the `select` command, followed by the device name:

```
(parted) select /dev/sdb
Using /dev/sdb
```

Getting Information

The print command can be used to get more information about a specific partition or even all of the block devices (disks) connected to your system.

To get information about the currently selected partition, just type print:

```
(parted) print
Model: ATA CT120BX500SSD1 (scsi)
Disk /dev/sda: 120GB
Sector size (logical/physical): 512B/512B
Partition Table: msdos
Disk Flags:
```

Number	Start	End	Size	Type	File system	Flags
1	2097kB	116GB	116GB	primary	ext4	
2	116GB	120GB	4295MB	primary	linux-swap(v1)	

You can get a list of all block devices connected to your system using print devices:

```
(parted) print devices
/dev/sdb (1999MB)
/dev/sda (120GB)
/dev/sdc (320GB)
/dev/mapper/cryptswap (4294MB)
```

To get information about all connected devices at once, you can use print all. If you wish to know how much free space there is in each one of them, you can use print free:

```
(parted) print free
Model: ATA CT120BX500SSD1 (scsi)
Disk /dev/sda: 120GB
Sector size (logical/physical): 512B/512B
Partition Table: msdos
Disk Flags:
```

Number	Start	End	Size	Type	File system	Flags
	32.3kB	2097kB	2065kB		Free Space	
1	2097kB	116GB	116GB	primary	ext4	
	116GB	116GB	512B		Free Space	
2	116GB	120GB	4295MB	primary	linux-swap(v1)	
	120GB	120GB	2098kB		Free Space	

Creating a Partition Table on an Empty Disk

Creating a Partition

To create a partition the command `mkpart` is used, using the syntax `mkpart PARTTYPE FSTYPE START END`, where:

PARTTYPE

Is the partition type, which can be primary, logical or extended in case an MBR partition table is used.

FSTYPE

Specifies which filesystem will be used on this partition. Note that parted will not create the filesystem. It just sets a flag on the partition which tells the OS what kind of data to expect from it.

START

Specifies the exact point on the device where the partition begins. You can use different units to specify this point. `2s` can be used to refer to the second sector of the disk, while `1m` refers to the beginning of the first megabyte of the disk. Other common units are `B` (bytes) and `%` (percentage of the disk).

END

Specifies the end of the partition. Note that this is not the size of the partition, this is the point on the disk where it ends. For example, if you specify `100m` the partition will end 100 MB after the start of the disk. You can use the same units as in the `START` parameter.

So, the command:

```
(parted) mkpart primary ext4 1m 100m
```

Creates a primary partition of type `ext4`, starting at the first megabyte of the disk, and ending after the 100th megabyte.

Removing a Partition

`parted` can recover a deleted partition. Consider you have the following partition structure:

Number	Start	End	Size	File system	Name	Flags
1	1049kB	99.6MB	98.6MB	ext4	primary	
2	99.6MB	200MB	100MB	ext4	primary	
3	200MB	300MB	99.6MB	ext4	primary	

By accident, you removed partition 2 using `rm 2`. To recover it, you can use the `rescue` command, with the syntax `rescue START END`, where `START` is the approximate location where the partition started, and `END` the approximate location where it ended.

`parted` will scan the disk in search of partitions, and offer to restore any that are found. In the example above the partition 2 started at 99,6 MB and ended at 200 MB. So you can use the following command to recover the partition:

```
(parted) rescue 90m 210m
```

Information: A ext4 primary partition was found at 99.6MB -> 200MB.

Do you want to add it to the partition table?

Yes/No/Cancel? y

This will recover the partition and its contents. Note that `rescue` can only recover partitions that have a filesystem installed on them. Empty partitions are not detected.

Resizing ext2/3/4 Partitions

parted can be used to resize partitions to make them bigger or smaller. However, there are some caveats:

During resizing the partition must be unused and unmounted.

You need enough free space after the partition to grow it to the size you want.

The command is `resizepart`, followed by the partition number and where it should end. For example, if you have the following partition table:

Number	Start	End	Size	File system	Name	Flags
1	1049kB	99.6MB	98.6MB	ext4	primary	
2	99.6MB	200MB	100MB	ext4		
3	200MB	300MB	99.6MB	ext4	primary	

Trying to grow partition 1 using `resizepart` would trigger an error message, because with the new size partition 1 would overlap with partition 2. However partition 3 can be resized as there is free space after it, which can be verified with the command `print free`:

```
(parted) print free
Model: Kingston DataTraveler 2.0 (scsi)
Disk /dev/sdb: 1999MB
Sector size (logical/physical): 512B/512B
Partition Table: gpt
Disk Flags:
```

Number	Start	End	Size	File system	Name	Flags
	17.4kB	1049kB	1031kB	Free Space		
1	1049kB	99.6MB	98.6MB	ext4	primary	
2	99.6MB	200MB	100MB	ext4		
3	200MB	300MB	99.6MB	ext4	primary	
	300MB	1999MB	1699MB	Free Space		

So you can use the following command to resize partition 3 to 350 MB:

```
(parted) resizepart 3 350m
```

```
(parted) print
Model: Kingston DataTraveler 2.0 (scsi)
Disk /dev/sdb: 1999MB
Sector size (logical/physical): 512B/512B
Partition Table: gpt
Disk Flags:
```

Number	Start	End	Size	File system	Name	Flags
1	1049kB	99.6MB	98.6MB	ext4	primary	
2	99.6MB	200MB	100MB	ext4		
3	200MB	350MB	150MB	ext4	primary	

Remember that the new end point is specified counting from the start of the disk. So, because partition 3 ended at 300 MB, now it needs to end at 350 MB.

But resizing the partition is only one part of the task. You also need to resize the filesystem that resides in it. For ext2/3/4 filesystems this is done with the `resize2fs` command. In the case of the example above, partition 3 still shows the “old” size when mounted:

```
$ df -h /dev/sdb3
```

```
Filesystem      Size  Used Avail Use% Mounted on
/dev/sdb3       88M  1.6M   80M   2% /media/carol/part3
```

To adjust the size the command `resize2fs` `DEVICE` `SIZE` can be used, where `DEVICE` corresponds to the partition you want to resize, and `SIZE` is the new size. If you omit the size parameter, it will use all of the available space of the partition. Before resizing, it is advised to unmount the partition.

In the example above:

```
$ sudo resize2fs /dev/sdb3
```

```
resize2fs 1.44.6 (5-Mar-2019)
```

```
Resizing the filesystem on /dev/sdb3 to 146212 (1k) blocks.
```

```
The filesystem on /dev/sdb3 is now 146212 (1k) blocks long.
```

```
$ df -h /dev/sdb3
```

```
Filesystem      Size  Used Avail Use% Mounted on
/dev/sdb3      135M  1.6M  123M   2% /media/carol/part3
```

To shrink a partition, the process needs to be done in the reverse order. First you resize the filesystem to the new, smaller size, then you resize the partition itself using `parted`.

Warning

Pay attention when shrinking partitions. If you get the order of things wrong, you will lose data!

In our example:

```
# resize2fs /dev/sdb3 88m
```

```
resize2fs 1.44.6 (5-Mar-2019)
```

```
Resizing the filesystem on /dev/sdb3 to 90112 (1k) blocks.
```

```
The filesystem on /dev/sdb3 is now 90112 (1k) blocks long.
```

```
# parted /dev/sdb3
```

```
(parted) resizepart 3 300m
```

```
Warning: Shrinking a partition can cause data loss, are you sure
you want to continue?
```

```
Yes/No? y
```

```
(parted) print
```

```
Model: Kingston DataTraveler 2.0 (scsi)
```

```
Disk /dev/sdb: 1999MB
```

```
Sector size (logical/physical): 512B/512B
```

```
Partition Table: gpt
```

```
Disk Flags:
```

```
Number  Start   End     Size    File system  Name    Flags
```

```
1      1049kB  99.6MB  98.6MB  ext4         primary
```

```
2      99.6MB  200MB   100MB   ext4
```

```
3      200MB   300MB   99.7MB  ext4         primary
```

Tip

Instead of specifying a new size, you can use the `-M` parameter of `resize2fs` to adjust the size of the filesystem so it is just big enough for the files on it.

Creating Swap Partitions

On Linux, the system can swap memory pages from RAM to disk as needed, storing them on a

separate space usually implemented as a separate partition on a disk, called the swap partition or simply swap. This partition needs to be of a specific type, and set-up with a proper utility (mkswap) before it can be used.

To create the swap partition using fdisk or gdisk, just proceed as if you were creating a regular partition, as explained before. The only difference is that you will need to change the partition type to Linux swap.

On fdisk use the t command. Select the partition you want to use and change its type to 82. Write changes to disk and quit with w.

On gdisk the command to change the partition type is also t, but the code is 8200. Write changes to disk and quit with w.

If you are using parted, the partition should be identified as a swap partition during creation, just use linux-swap as the filesystem type. For example, the command to create a 500 MB swap partition, starting at 300 MB on disk is:

```
(parted) mkpart primary linux-swap 301m 800m
```

Once the partition is created and properly identified, just use mkswap followed by the device representing the partition you want to use, like:

```
# mkswap /dev/sda2
```

To enable swap on this partition, use swapon followed by the device name:

```
# swapon /dev/sda2
```

Similarly, swapoff, followed by the device name, will disable swap on that device.

Linux also supports the use of swap files instead of partitions. Just create an empty file of the size you want using dd and then use mkswap and swapon with this file as the target.

The following commands will create a 1 GB file called myswap in the current directory, filled with zeroes, and then set-up and enable it as a swap file.

Create the swap file:

```
$ dd if=/dev/zero of=myswap bs=1M count=1024
```

```
1024+0 records in
```

```
1024+0 records out
```

```
1073741824 bytes (1.1 GB, 1.0 GiB) copied, 7.49254 s, 143 MB/s
```

if= is the input file, the source of the data that will be written to the file. In this case it is the /dev/zero device, which provides as many NULL characters as requested. of= is the output file, the file that will be created. bs= is the size of the data blocks, here specified in Megabytes, and count= is the amount of blocks to be written to the output. 1024 blocks of 1 MB each equals 1 GB.

```
# mkswap myswap
```

```
Setting up swapspace version 1, size = 1024 MiB (1073737728 bytes)
```

```
no label, UUID=49c53bc4-c4b1-4a8b-a613-8f42cb275b2b
```

```
# swapon myswap
```

Using the commands above, this swap file will be used only during the current system session. If the machine is rebooted, the file will still be available, but will not be automatically loaded. You can automate that by adding the new swap file to /etc/fstab, which we will discuss in a later lesson.

Tip

Both mkswap and swapon will complain if your swap file has insecure permissions. The recommended file permission flag is 0600. Owner and group should be root.

Maintain the integrity of filesystems

Modern Linux filesystems are journaled. This means that every operation is registered in an internal log (the journal) before it is executed. If the operation is interrupted due to a system error (like a kernel panic, power failure, etc.) it can be reconstructed by checking the journal, avoiding filesystem corruption and loss of data.

This greatly reduces the need for manual filesystem checks, but they may still be needed. Knowing the tools used for this (and the corresponding parameters) may represent the difference between dinner at home with your family or an all-nighter in the server room at work.

In this lesson, we will discuss the tools available to monitor filesystem usage, optimize its operation and how to check and repair damage.

Checking Disk Usage

There are two commands that can be used to check how much space is being used and how much is left on a filesystem. The first one is `du`, which stands for “disk usage”.

`du` is recursive in nature. In its most basic form, the command will simply show how many 1 Kilobyte blocks are being used by the current directory and all its subdirectories:

```
$ du
4816 .
```

This is not very helpful, so we can request more “human readable” output by adding the `-h` parameter:

```
$ du -h
4.8M .
```

By default, `du` only shows the usage count for directories (considering all files and subdirectories inside it). To show an individual count for all files in the directory, use the `-a` parameter:

```
$ du -ah
432K  ./geminoid.jpg
508K  ./Linear_B_Hero.jpg
468K  ./LG-G8S-ThinQ-Mirror-White.jpg
656K  ./LG-G8S-ThinQ-Range.jpg
60K   ./Stranger3_Titulo.png
108K  ./Baidu_Banho.jpg
324K  ./Xiaomi_Mimoji.png
284K  ./Mi_CC_9e.jpg
96K   ./Mimoji_Comparativo.jpg
32K   ./Xiaomi FCC.jpg
484K  ./geminoid2.jpg
108K  ./Mimoji_Abre.jpg
88K   ./Mi8_Hero.jpg
832K  ./Tablet_Linear_B.jpg
332K  ./Mimoji_Comparativo.png
4.8M  .
```

The default behaviour is to show the usage of every subdirectory, then the total usage of the current directory, including subdirectories:

```
$ du -h
4.8M    ./Temp
6.0M    .
```

In the example above, we can see that the subdirectory Temp occupies 4.8 MB and the current directory, including Temp, occupies 6.0 MB. But how much space do the files in the current directory occupy, excluding the subdirectories? For that we have the -S parameter:

```
$ du -Sh
4.8M    ./Temp
1.3M    .
```

Tip

Keep in mind that command line parameters are case-sensitive: -s is different from -S.

If you want to keep this distinction between the space used by the files in the current directory and the space used by subdirectories, but also want a grand total at the end, you can add the -c parameter:

```
$ du -Shc
4.8M    ./Temp
1.3M    .
6.0M    total
```

You can control how “deep” the output of du should go with the -d N parameter, where N describes the levels. For example, if you use the -d 1 parameter, it will show the current directory and its subdirectories, but not the subdirectories of those.

See the difference below. Without -d:

```
$ du -h
216K    ./somedir/anotherdir
224K    ./somedir
232K    .
```

And limiting the depth to one level with -d 1:

```
$ du -h -d1
224K    ./somedir
232K    .
```

Please note that even if anotherdir is not being shown, its size is still being taken into account.

You may wish to exclude some types of files from the count, which you can do with --exclude="PATTERN", where PATTERN is the pattern against which you wish to match. Consider this directory:

```
$ du -ah
124K    ./ASM68K.EXE
2.0M    ./Contra.bin
36K    ./fixheadr.exe
4.0K    ./README.txt
2.1M    ./Contra_NEW.bin
4.0K    ./Built.bat
8.0K    ./Contra_Main.asm
4.2M    .
```

Now, we will use --exclude to filter out every file with the .bin extension:

```
$ du -ah --exclude="*.bin"
124K    ./ASM68K.EXE
36K    ./fixheadr.exe
4.0K    ./README.txt
```

```
4.0K    ./Built.bat
8.0K    ./Contra_Main.asm
180K    .
```

Note that the total no longer reflects the size of the excluded files.

Checking for Free Space

`du` works at the files level. There is another command that can show you disk usage, and how much space is available, at the filesystem level. This command is `df`.

The command `df` will provide a list of all of the available (already mounted) filesystems on your system, including their total size, how much space has been used, how much space is available, the usage percentage and where it is mounted:

```
$ df
Filesystem 1K-blocks  Used Available Use% Mounted on
udev       2943068      0 2943068 0% /dev
tmpfs      595892    2496 593396 1% /run
/dev/sda1  110722904 25600600 79454800 25% /
tmpfs      2979440  951208 2028232 32% /dev/shm
tmpfs      5120      0    5120 0% /run/lock
tmpfs      2979440      0 2979440 0% /sys/fs/cgroup
tmpfs      595888     24 595864 1% /run/user/119
tmpfs      595888    116 595772 1% /run/user/1000
/dev/sdb1   89111     1550 80824 2% /media/carol/part1
/dev/sdb3   83187     1550 75330 3% /media/carol/part3
/dev/sdb2   90827     1921 82045 3% /media/carol/part2
/dev/sdc1  312570036 233740356 78829680 75% /media/carol/Samsung Externo
```

However, showing the size in 1 KB blocks is not very user-friendly. Like on `du`, you can add the `-h` parameters to get a more “human readable” output:

```
$ df -h
Filesystem  Size  Used Avail Use% Mounted on
udev        2.9G   0 2.9G   0% /dev
tmpfs       582M 2.5M 580M   1% /run
/dev/sda1   106G 25G 76G   25% /
tmpfs       2.9G 930M 2.0G  32% /dev/shm
tmpfs       5.0M   0 5.0M   0% /run/lock
tmpfs       2.9G   0 2.9G   0% /sys/fs/cgroup
tmpfs       582M 24K 582M   1% /run/user/119
tmpfs       582M 116K 582M   1% /run/user/1000
/dev/sdb1   88M 1.6M 79M   2% /media/carol/part1
/dev/sdb3   82M 1.6M 74M   3% /media/carol/part3
/dev/sdb2   89M 1.9M 81M   3% /media/carol/part2
/dev/sdc1   299G 223G 76G  75% /media/carol/Samsung Externo
```

You can also use the `-i` parameter to show used/available inodes instead of blocks:

```
$ df -i
Filesystem  Inodes IUsed IFree IUse% Mounted on
udev       737142  547 736595   1% /dev
tmpfs      745218  908 744310   1% /run
/dev/sda6  6766592 307153 6459439   5% /
tmpfs      745218  215 745003   1% /dev/shm
tmpfs      745218    4 745214   1% /run/lock
tmpfs      745218   18 745200   1% /sys/fs/cgroup
```

```
/dev/sda1    62464  355 62109  1% /boot
tmpfs       745218  43 745175  1% /run/user/1000
```

One useful parameter is -T, which will also print the type of each filesystem:

```
$ df -hT
```

Filesystem	Type	Size	Used	Avail	Use%	Mounted on
udev	devtmpfs	2.9G	0	2.9G	0%	/dev
tmpfs	tmpfs	582M	2.5M	580M	1%	/run
/dev/sda1	ext4	106G	25G	76G	25%	/
tmpfs	tmpfs	2.9G	930M	2.0G	32%	/dev/shm
tmpfs	tmpfs	5.0M	0	5.0M	0%	/run/lock
tmpfs	tmpfs	2.9G	0	2.9G	0%	/sys/fs/cgroup
tmpfs	tmpfs	582M	24K	582M	1%	/run/user/119
tmpfs	tmpfs	582M	116K	582M	1%	/run/user/1000
/dev/sdb1	ext4	88M	1.6M	79M	2%	/media/carol/part1
/dev/sdb3	ext4	82M	1.6M	74M	3%	/media/carol/part3
/dev/sdb2	ext4	89M	1.9M	81M	3%	/media/carol/part2
/dev/sdc1	fuseblk	299G	223G	76G	75%	/media/carol/Samsung Externo

Knowing the type of the filesystem you can filter the output. You can show only filesystems of a given type with -t TYPE, or exclude filesystems of a given type with -x TYPE, like in the examples below.

Excluding tmpfs filesystems:

```
$ df -hx tmpfs
```

Filesystem	Size	Used	Avail	Use%	Mounted on
udev	2.9G	0	2.9G	0%	/dev
/dev/sda1	106G	25G	76G	25%	/
/dev/sdb1	88M	1.6M	79M	2%	/media/carol/part1
/dev/sdb3	82M	1.6M	74M	3%	/media/carol/part3
/dev/sdb2	89M	1.9M	81M	3%	/media/carol/part2
/dev/sdc1	299G	223G	76G	75%	/media/carol/Samsung Externo

Showing only ext4 filesystems:

```
$ df -ht ext4
```

Filesystem	Size	Used	Avail	Use%	Mounted on
/dev/sda1	106G	25G	76G	25%	/
/dev/sdb1	88M	1.6M	79M	2%	/media/carol/part1
/dev/sdb3	82M	1.6M	74M	3%	/media/carol/part3
/dev/sdb2	89M	1.9M	81M	3%	/media/carol/part2

You can also customize the output of df, selecting what should be displayed and in which order, using the --output= parameter followed by a comma separated list of fields you wish to display. Some of the available fields are:

source

The device corresponding to the filesystem.

fstype

The filesystem type.

size

The total size of the filesystem.

used

How much space is being used.

avail

How much space is available.

pcent

The usage percentage.

target

Where the filesystem is mounted (mount point).

If you want an output showing the target, source, type and usage, you can use:

```
$ df -h --output=target,source,fstype,pcent
```

Mounted on	Filesystem	Type	Use%
/dev	udev	devtmpfs	0%
/run	tmpfs	tmpfs	1%
/	/dev/sda1	ext4	25%
/dev/shm	tmpfs	tmpfs	32%
/run/lock	tmpfs	tmpfs	0%
/sys/fs/cgroup	tmpfs	tmpfs	0%
/run/user/119	tmpfs	tmpfs	1%
/run/user/1000	tmpfs	tmpfs	1%
/media/carol/part1	/dev/sdb1	ext4	2%
/media/carol/part3	/dev/sdb3	ext4	3%
/media/carol/part2	/dev/sdb2	ext4	3%
/media/carol/Samsung Externo	/dev/sdc1	fuseblk	75%

df can also be used to check inode information, by passing the following fields to --output=:

itotal

The total number of inodes in the filesystem.

iused

The number of used inodes in the filesystem.

iavail

The number of available inodes in the filesystem.

ipcent

The percentage of used inodes in the filesystem.

For example:

```
$ df --output=source,fstype,itotal,iused,ipcent
```

Filesystem	Type	Inodes	IUsed	IUse%
udev	devtmpfs	735764	593	1%
tmpfs	tmpfs	744858	1048	1%
/dev/sda1	ext4	7069696	318651	5%
tmpfs	tmpfs	744858	222	1%
tmpfs	tmpfs	744858	3	1%
tmpfs	tmpfs	744858	18	1%
tmpfs	tmpfs	744858	22	1%
tmpfs	tmpfs	744858	40	1%

Maintaining ext2, ext3 and ext4 Filesystems

To check a filesystem for errors (and hopefully fix them), Linux provides the fsck utility (think of “filesystem check” and you will never forget the name). In its most basic form, you can invoke it with

fsck followed by the filesystem's location you want to check:

```
# fsck /dev/sdb1
fsck from util-linux 2.33.1
e2fsck 1.44.6 (5-Mar-2019)
DT_2GB: clean, 20/121920 files, 369880/487680 blocks
Warning
NEVER run fsck (or related utilities) on a mounted filesystem. If this is done anyway, data may be lost.
```

fsck itself will not check the filesystem, it will merely call the appropriate utility for the filesystem type to do so. In the example above, since a filesystem type was not specified, fsck assumed an ext2/3/4 filesystem by default, and called e2fsck.

To specify a filesystem, use the -t option, followed by the filesystem name, like in `fsck -t vfat /dev/sdc`. Alternatively, you may call a filesystem-specific utility directly, like `fsck.msdos` for FAT filesystems.

Tip

Type `fsck` followed by Tab twice to see a list of all related commands on your system.

fsck can take some command-line arguments. These are some of the most common:

-A

This will check all filesystems listed in `/etc/fstab`.

-C

Displays a progress bar when checking a filesystem. Currently only works on ext2/3/4 filesystems.

-N

This will print what would be done and exit, without actually checking the filesystem.

-R

When used in conjunction with -A, this will skip checking the root filesystem.

-V

Verbose mode, prints more information than usual during operation. This is useful for debugging.

The specific utility for ext2, ext3 and ext4 filesystems is e2fsck, also called `fsck.ext2`, `fsck.ext3` and `fsck.ext4` (those three are merely links to e2fsck). By default, it runs in interactive mode: when a filesystem error is found, it stops and asks the user what to do. The user must type y to fix the problem, n to leave it unfixed or a to fix the current problem and all subsequent ones.

Of course sitting in front of a terminal waiting for e2fsck to ask what to do is not a productive use of your time, especially if you are dealing with a big filesystem. So, there are options that cause e2fsck to run in non-interactive mode:

-p

This will attempt to automatically fix any errors found. If an error that requires intervention from the system administrator is found, e2fsck will provide a description of the problem and exit.

-y

This will answer y (yes) to all questions.

-n

The opposite of -y. Besides answering n (no) to all questions, this will cause the filesystem to be mounted read-only, so it cannot be modified.

-f

Forces e2fsck to check a filesystem even if is marked as “clean”, i.e. has been correctly unmounted.

Fine Tuning an ext Filesystem

ext2, ext3 and ext4 filesystems have a number of parameters that can be adjusted, or “tuned”, by the system administrator to better suit the system needs. The utility used to display or modify these parameters is called tune2fs.

To see the current parameters for any given filesystem, use the -l parameter followed by the device representing the partition. The example below shows the output of this command on the first partition of the first disk (/dev/sda1) of a machine:

```
# tune2fs -l /dev/sda1
tune2fs 1.44.6 (5-Mar-2019)
Filesystem volume name: <none>
Last mounted on:      /
Filesystem UUID:      6e2c12e3-472d-4bac-a257-c49ac07f3761
Filesystem magic number: 0xEF53
Filesystem revision #: 1 (dynamic)
Filesystem features:   has_journal ext_attr resize_inode dir_index filetype needs_recovery extent
64bit flex_bg sparse_super large_file huge_file dir_nlink extra_isize metadata_csum
Filesystem flags:      signed_directory_hash
Default mount options: user_xattr acl
Filesystem state:      clean
Errors behavior:       Continue
Filesystem OS type:    Linux
Inode count:           7069696
Block count:           28255605
Reserved block count:  1412780
Free blocks:           23007462
Free inodes:           6801648
First block:           0
Block size:            4096
Fragment size:         4096
Group descriptor size:  64
Reserved GDT blocks:   1024
Blocks per group:      32768
Fragments per group:   32768
Inodes per group:      8192
Inode blocks per group: 512
Flex block group size: 16
Filesystem created:    Mon Jun 17 13:49:59 2019
Last mount time:       Fri Jun 28 21:14:38 2019
Last write time:       Mon Jun 17 13:53:39 2019
Mount count:           8
Maximum mount count:   -1
Last checked:          Mon Jun 17 13:49:59 2019
Check interval:        0 (<none>)
Lifetime writes:       20 GB
Reserved blocks uid:   0 (user root)
Reserved blocks gid:   0 (group root)
First inode:           11
Inode size:            256
```

Required extra isize: 32
Desired extra isize: 32
Journal inode: 8
First orphan inode: 5117383
Default directory hash: half_md4
Directory Hash Seed: fa95a22a-a119-4667-a73e-78f77af6172f
Journal backup: inode blocks
Checksum type: crc32c
Checksum: 0xe084fe23

ext filesystems have mount counts. The count is increased by 1 each time the filesystem is mounted, and when a threshold value (the maximum mount count) is reached the system will be automatically checked with `e2fsck` on the next boot.

The maximum mount count can be set with the `-c N` parameter, where `N` is the number of times the filesystem can be mounted without being checked. The `-C N` parameter sets the number of times the system has been mounted to the value of `N`. Note that command line parameters are case-sensitive, so `-c` is different from `-C`.

It is also possible to define a time interval between checks, with the `-i` parameter, followed by a number and the letters `d` for days, `m` for months and `y` for years. For example, `-i 10d` would check the filesystem at the next reboot every 10 days. Use zero as the value to disable this feature.

`-L` can be used to set a label for the filesystem. This label can have up to 16 characters. The `-U` parameter sets the UUID for the filesystem, which is a 128 bit hexadecimal number. In the example above, the UUID is `6e2c12e3-472d-4bac-a257-c49ac07f3761`. Both the label and UUID can be used instead of the device name (like `/dev/sda1`) to mount the filesystem.

The option `-e BEHAVIOUR` defines the kernel behaviour when a filesystem error is found. There are three possible behaviours:

`continue`

Will continue execution normally.

`remount-ro`

Will remount the filesystem as read-only.

`panic`

Will cause a kernel panic.

The default behaviour is to continue. `remount-ro` might be useful in data-sensitive applications, as it will immediately stop writes to the disk, avoiding more potential errors.

ext3 filesystems are basically ext2 filesystems with a journal. Using `tune2fs` you can add a journal to an ext2 filesystem, thus converting it to ext3. The procedure is simple, just pass the `-j` parameter to `tune2fs`, followed by the device containing the filesystem:

```
# tune2fs -j /dev/sda1
```

Afterwards, when mounting the converted filesystem, do not forget to set the type to ext3 so the journal can be used.

When dealing with journaled filesystems, the `-J` parameter allows you to use extra parameters to set some journal options, like `-J size=` to set the journal size (in megabytes), `-J location=` to specify where the journal should be stored (either a specific block, or a specific position on the disk with suffixes like `M` or `G`) and even put the journal on an external device with `-J device=`.

You can specify multiple parameters at once by separating them with a comma. For example: `-J size=10,location=100M,device=/dev/sdb1` will create a 10 MB journal at the 100 MB position on the

device /dev/sdb1.

Warning

tune2fs has a “brute force” option, -f, which will force it to complete an operation even if errors are found. Needless to say, this should be only used with extreme caution.

Maintaining XFS Filesystems

For XFS filesystems, the equivalent of fsck is xfs_repair. If you suspect that something is wrong with the filesystem, the first thing to do is to scan it for damage.

This can be done by passing the -n parameter to xfs_repair, followed by the device containing the filesystem. The -n parameter means “no modify”: the filesystem will be checked, errors will be reported but no repairs will be made:

```
# xfs_repair -n /dev/sdb1
```

Phase 1 - find and verify superblock...

Phase 2 - using internal log

- zero log...
- scan filesystem freespace and inode maps...
- found root inode chunk

Phase 3 - for each AG...

- scan (but do not clear) agi unlinked lists...
- process known inodes and perform inode discovery...
- agno = 0
- agno = 1
- agno = 2
- agno = 3
- process newly discovered inodes...

Phase 4 - check for duplicate blocks...

- setting up duplicate extent list...
- check for inodes claiming duplicate blocks...
- agno = 1
- agno = 3
- agno = 0
- agno = 2

No modify flag set, skipping phase 5

Phase 6 - check inode connectivity...

- traversing filesystem ...
- traversal finished ...
- moving disconnected inodes to lost+found ...

Phase 7 - verify link counts...

No modify flag set, skipping filesystem flush and exiting.

If errors are found, you can proceed to do the repairs without the -n parameter, like so: xfs_repair /dev/sdb1.

xfs_repair accepts a number of command line options. Among them:

-l LOGDEV and -r RTDEV

These are needed if the filesystem has external log and realtime sections. In this case, replace LOGDEV and RTDEV with the corresponding devices.

-m N

Is used to limit the memory usage of xfs_repair to N megabytes, something which can be useful on server settings. According to the man page, by default xfs_repair will scale its memory usage as

needed, up to 75% of the system's physical RAM.

-d

The “dangerous” mode will enable the repair of filesystems that are mounted read-only.

-v

You may have guessed it: verbose mode. Each time this parameter is used, the “verbosity” is increased (for example, -v -v will print more information than just -v).

Note that `xfs_repair` is unable to repair filesystems with a “dirty” log. You can “zero out” a corrupt log with the -L parameter, but keep in mind that this is a last resort as it may result in filesystem corruption and data loss.

To debug an XFS filesystem, the utility `xfs_db` can be used, like in `xfs_db /dev/sdb1`. This is mostly used to inspect various elements and parameters of the filesystem.

This utility has an interactive prompt, like `parted`, with many internal commands. A help system is also available: type `help` to see a list of all commands, and `help` followed by the command name to see more information about the command.

Another useful utility is `xfs_fsr`, which can be used to reorganize (“defragment”) an XFS filesystem. When executed without any extra arguments it will run for two hours and try to defragment all mounted, writable XFS filesystems listed on the `/etc/mtab/` file. You may need to install this utility using the package manager for your Linux distribution, as it may not be part of a default install. For more information consult the corresponding man page.

Control mounting and unmounting of filesystems

Introduction

Up until now, you learned how to partition disks and how to create and maintain filesystems on them. However 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. Filesystems can be mounted manually or automatically and there are many ways to do this. We will learn about some of them in this lesson.

Mounting and Unmounting Filesystems

The command to manually mount a filesystem is called `mount` and its syntax is:

`mount -t TYPE DEVICE MOUNTPOINT`

Where:

TYPE

The type of the filesystem being mounted (e.g. `ext4`, `btrfs`, `exfat`, etc.).

DEVICE

The name of the partition containing the filesystem (e.g. `/dev/sdb1`)

MOUNTPOINT

Where the filesystem will be mounted. The mounted-on directory need not be empty, although it must exist. Any files in it, however, will be inaccessible by name while the filesystem is mounted.

For example, to mount a USB flash drive containing an exFAT filesystem located on /dev/sdb1 to a directory called flash under your home directory, you could use:

```
# mount -t exfat /dev/sdb1 ~/flash/
```

Tip

Many Linux systems use the Bash shell, and on those the tilde ~ on the path to the mountpoint is a shorthand for the current user's home directory. If the current user is named john, for example, it will be replaced by /home/john.

After mounting, the contents of the filesystem will be accessible under the ~/flash directory:

```
$ ls -lh ~/flash/
total 469M
-rwxrwxrwx 1 root root 454M jul 19 09:49 lineage-16.0-20190711-MOD-quark.zip
-rwxrwxrwx 1 root root 16M jul 19 09:44 twrp-3.2.3-mod_4-quark.img
```

Listing Mounted Filesystems

If you type just mount, you will get a list of all the filesystems currently mounted on your system. This list can be quite large, because in addition to the disks attached to your system, it also contains a number of run-time filesystems in memory that serve various purposes. To filter the output, you can use the -t parameter to list only filesystems of the corresponding type, like below:

```
# mount -t ext4
/dev/sda1 on / type ext4 (rw,noatime,errors=remount-ro)
```

You can specify multiple filesystems at once by separating them with a comma:

```
# mount -t ext4,fuseblk
/dev/sda1 on / type ext4 (rw,noatime,errors=remount-ro)
/dev/sdb1 on /home/carol/flash type fuseblk
(rw,nosuid,nodev,relatime,user_id=0,group_id=0,default_permissions,allow_other,blksize=4096)
[DT_8GB]
```

The output in the examples above can be described in the format:

SOURCE on TARGET type TYPE OPTIONS

Where SOURCE is the partition which contains the filesystem, TARGET is the directory where it is mounted, TYPE is the filesystem type and OPTIONS are the options passed to the mount command at mount time.

Additional Command Line Parameters

There are many command line parameters that can be used with mount. Some of the most used ones are:

-a

This will mount all filesystems listed in the file /etc/fstab (more on that in the next section).

-o or --options

This will pass a list of comma-separated mount options to the mount command, which can change how the filesystem will be mounted. These will also be discussed alongside /etc/fstab.

-r or -ro

This will mount the filesystem as read-only.

-w or -rw

This will mount the filesystem as writable.

To unmount a filesystem, use the `umount` command, followed by the device name or the mount point. Considering the example above, the commands below are interchangeable:

```
# umount /dev/sdb1
```

```
# umount ~/flash
```

Some of the command line parameters to `umount` are:

`-a`

This will unmount all filesystems listed in `/etc/fstab`.

`-f`

This will force the unmounting of a filesystem. This may be useful if you mounted a remote filesystem that has become unreachable.

`-r`

If the filesystem cannot be unmounted, this will try to make it read-only.

Dealing with Open Files

When unmounting a filesystem, you may encounter an error message stating that the target is busy. This will happen if any files on the filesystem are open. However, it may not be immediately obvious where an open file is located, or what is accessing the filesystem.

In such cases you may use the `lsof` command, followed by the name of the device containing the filesystem, to see a list of processes accessing it and which files are open. For example:

```
# umount /dev/sdb1
```

```
umount: /media/carol/External_Drive: target is busy.
```

```
# lsof /dev/sdb1
```

```
COMMAND PID  USER  FD  TYPE DEVICE SIZE/OFF NODE NAME
```

```
evince 3135 carol 16r  REG  8,17 21881768 5195 /media/carol/External_Drive/Documents/E-Books/MagPi40.pdf
```

`COMMAND` is the name of the executable that opened the file, and `PID` is the process number. `NAME` is the name of the file that is open. In the example above, the file `MagPi40.pdf` is opened by the program `evince` (a PDF viewer). If we close the program, then we will be able to unmount the filesystem.

Note

Before the `lsof` output appears GNOME users may see a warning message in the terminal window.

```
lsof: WARNING: can't stat() fuse.gvfsd-fuse file system /run/user/1000/gvfs
```

Output information may be incomplete.

`lsof` tries to process all mounted filesystems. This warning message is raised because `lsof` has encountered a GNOME Virtual file system (GVFS). This is a special case of a filesystem in user space (FUSE). It acts as a bridge between GNOME, its APIs and the kernel. No one—not even root—can access one of these file systems, apart from the owner who mounted it (in this case, GNOME). You can ignore this warning.

Where to Mount?

You can mount a filesystem 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, 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.

Mounting Filesystems on Bootup

The file /etc/fstab contains descriptions about the filesystems that can be mounted. This is a text file, where each line describes a filesystem to be mounted, with six fields per line in the following order:

FILESYSTEM MOUNTPOINT TYPE OPTIONS DUMP PASS

Where:

FILESYSTEM

The device containing the filesystem to be mounted. Instead of the device, you can specify the UUID or label of the partition, something which we will discuss later on.

MOUNTPOINT

Where the filesystem will be mounted.

TYPE

The filesystem type.

OPTIONS

Mount options that will be passed to mount.

DUMP

Indicates whether any ext2, ext3 or ext4 filesystems should be considered for backup by the dump command. Usually it is zero, meaning they should be ignored.

PASS

When non-zero, defines the order in which the filesystems will be checked on bootup. Usually it is zero.

For example, the first partition on the first disk of a machine could be described as:

```
/dev/sda1 / ext4 noatime,errors
```

The mount options on OPTIONS are a comma-separated list of parameters, which can be generic or filesystem specific. Among the generic ones we have:

atime and noatime

By default, every time a file is read the access time information is updated. Disabling this (with noatime) can speed up disk I/O. Do not confuse this with the modification time, which is updated every time a file is written to.

auto and noauto

Whether the filesystem can (or can not) be mounted automatically with mount -a.

defaults

This will pass the options rw, suid, dev, exec, auto, nouser and async to mount.

dev and nodev

Whether character or block devices in the mounted filesystem should be interpreted.

exec and noexec

Allow or deny permission to execute binaries on the filesystem.

user and nouser

Allows (or not) an ordinary user to mount the filesystem.

group

Allows a user to mount the filesystem if the user belongs to the same group which owns the device containing it.

owner

Allows a user to mount a filesystem if the user owns the device containing it.

suid and nosuid

Allow, or not, SETUID and SETGID bits to take effect.

ro and rw

Mount a filesystem as read-only or writable.

remount

This will attempt to remount an already mounted filesystem. This is not used on /etc/fstab, but as a parameter to mount -o. For example, to remount the already mounted partition /dev/sdb1 as read-only, you could use the command mount -o remount,ro /dev/sdb1. When remounting, you do not need to specify the filesystem type, only the device name or the mount point.

sync and async

Whether to do all I/O operations to the filesystem synchronously or asynchronously. async is usually the default. The manual page for mount warns that using sync on media with a limited number of write cycles (like flash drives or memory cards) may shorten the life span of the device.

Using UUIDs and Labels

Specifying the name of the device containing the filesystem to mount may introduce some problems. Sometimes the same device name may be assigned to another device depending on when, or where, it was connected to your system. For example, a USB flash drive on `/dev/sdb1` may be assigned to `/dev/sdc1` if plugged on another port, or after another flash drive.

One way to avoid this is to specify the label or UUID (*Universally Unique Identifier*) of the volume. Both are specified when the filesystem is created and will not change, unless the filesystem is destroyed or manually assigned a new label or UUID.

label or UUID.

The command `lsblk` can be used to query information about a filesystem and find out the label and UUID associated to it. To do this, use the `-f` parameter, followed by the device name:

```
$ lsblk -f /dev/sda1
```

NAME	FSTYPE	LABEL	UUID	FSAVAIL	FSUSE%	MOUNTPOINT
sda1	ext4		6e2c12e3-472d-4bac-a257-c49ac07f3761	64,9G	33%	/

Here is the meaning of each column:

NAME

Name of the device containing the filesystem.

FSTYPE

Filesystem type.

LABEL

Filesystem label.

UUID

Universally Unique Identifier (UUID) assigned to the filesystem.

FSAVAIL

How much space is available in the filesystem.

FSUSE%

Usage percentage of the filesystem.

MOUNTPOINT

Where the filesystem is mounted.

In `/etc/fstab` a device can be specified by its UUID with the `UUID=` option, followed by the UUID, or with `LABEL=`, followed by the label. So, instead of:

```
/dev/sda1 / ext4 noatime,errors
```

You would use:

```
UUID=6e2c12e3-472d-4bac-a257-c49ac07f3761 / ext4 noatime,errors
```

Or, if you have a disk labeled `homedisk`:

```
LABEL=homedisk /home ext4 defaults
```

The same syntax can be used with the `mount` command. Instead of the device name, pass the UUID or label. For example, to mount an external NTFS disk with the UUID `56C11DCC5D2E1334` on `/mnt/external`, the command would be:

```
$ mount -t ntfs UUID=56C11DCC5D2E1334 /mnt/external
```

Mounting Disks with Systemd

Systemd is the init process, the first process to run on many Linux distributions. It is responsible for spawning other processes, starting services and bootstrapping the system. Among many other tasks, systemd can also be used to manage the mounting (and automounting) of filesystems.

To use this feature of systemd, you need to create a configuration file called a mount unit. Each volume to be mounted gets its own mount unit and they need to be placed in `/etc/systemd/system/`.

Mount units are simple text files with the `.mount` extension. The basic format is shown below:

```
[Unit]
Description=
```

```
[Mount]
What=
Where=
Type=
Options=
```

```
[Install]
WantedBy=
Description=
Short description of the mount unit, something like Mounts the backup disk.
```

What=
What should be mounted. The volume must be specified as `/dev/disk/by-uuid/VOL_UUID` where `VOL_UUID` is the UUID of the volume.

Where=
Should be the full path to where the volume should be mounted.

Type=
The filesystem type.

Options=
Mount options that you may wish to pass, these are the same used with the `mount` command or in `/etc/fstab`.

WantedBy=
Used for dependency management. In this case, we will use `multi-user.target`, which means that whenever the system boots into a multi-user environment (a normal boot) the unit will be mounted.

Our previous example of the external disk could be written as:

```
[Unit]
Description=External data disk

[Mount]
What=/dev/disk/by-uuid/56C11DCC5D2E1334
Where=/mnt/external
Type=ntfs
Options=defaults
```

```
[Install]
WantedBy=multi-user.target
```

But we are not done yet. To work correctly, the mount unit must have the same name as the mount point. In this case, the mount point is `/mnt/external`, so the file should be named `mnt-external.mount`.

After that, you need to restart the systemd daemon with the systemctl command, and start the unit:

```
# systemctl daemon-reload
```

```
# systemctl start mnt-external.mount
```

Now the contents of the external disk should be available on /mnt/external. You can check the status of the mounting with the command systemctl status mnt-external.mount, like below:

```
# systemctl status mnt-external.mount
```

```
● mnt-external.mount - External data disk
```

```
Loaded: loaded (/etc/systemd/system/mnt-external.mount; disabled; vendor pres
```

```
Active: active (mounted) since Mon 2019-08-19 22:27:02 -03; 14s ago
```

```
Where: /mnt/external
```

```
What: /dev/sdb1
```

```
Tasks: 0 (limit: 4915)
```

```
Memory: 128.0K
```

```
CGroup: /system.slice/mnt-external.mount
```

```
ago 19 22:27:02 pop-os systemd[1]: Mounting External data disk...
```

```
ago 19 22:27:02 pop-os systemd[1]: Mounted External data disk.
```

The systemctl start mnt-external.mount command will only enable the unit for the current session. If you want to enable it on every boot, replace start with enable:

```
# systemctl enable mnt-external.mount
```

Automounting a Mount Unit

Mount units can be automounted whenever the mount point is accessed. To do this, you need an .automount file, alongside the .mount file describing the unit. The basic format is:

```
[Unit]
```

```
Description=
```

```
[Automount]
```

```
Where=
```

```
[Install]
```

```
WantedBy=multi-user.target
```

Like before, Description= is a short description of the file, and Where= is the mountpoint. For example, an .automount file for our previous example would be:

```
[Unit]
```

```
Description=Automount for the external data disk
```

```
[Automount]
```

```
Where=/mnt/external
```

```
[Install]
```

```
WantedBy=multi-user.target
```

Save the file with the same name as the mount point (in this case, mnt-external.automount), reload systemd and start the unit:

```
# systemctl daemon-reload
```

```
# systemctl start mnt-external.automount
```

Now whenever the /mnt/external directory is accessed, the disk will be mounted. Like before, to

enable the automount on every boot you would use:

```
# systemctl enable mnt-external.automount
```

Manage file permissions and ownership

Introduction

Being a multi-user system, Linux needs some way to track who owns each file and whether or not a user is allowed to perform actions on a file. This is to ensure the privacy of users who might want to keep the contents of their files confidential, as well as to ensure collaboration by making certain files accessible to multiple users.

This is done through a three-level permissions system. Every file on disk is owned by a user and a user group and has three sets of permissions: one for its owner, one for the group who owns the file and one for everyone else. In this lesson, you will learn how to query the permissions for a file, the meaning of these permissions and how to manipulate them.

Querying Information about Files and Directories

The command `ls` is used to get a listing of the contents of any directory. In this basic form, all you get are the filenames:

```
$ ls
```

```
Another_Directory picture.jpg text.txt
```

But there is much more information available for each file, including its type, size, ownership and more. To see this information you must ask `ls` for a “long form” listing, using the `-l` parameter:

```
$ ls -l
```

```
total 536
```

```
drwxrwxr-x 2 carol carol 4096 Dec 10 15:57 Another_Directory
```

```
-rw----- 1 carol carol 539663 Dec 10 10:43 picture.jpg
```

```
-rw-rw-r-- 1 carol carol 1881 Dec 10 15:57 text.txt
```

Each column on the output above has a meaning. Let us have a look at the columns relevant for this lesson.

The first column on the listing shows the file type and permissions. For example, on `drwxrwxr-x`:

The first character, `d`, indicates the file type.

The next three characters, `rw`, indicate the permissions for the owner of the file, also referred to as user or `u`.

The next three characters, `rw`, indicate the permissions of the group owning the file, also referred to as `g`.

The last three characters, `-x`, indicate the permissions for anyone else, also known as others or `o`.

Tip

It is also common to hear the others permission set referred to as world permissions, as in “Everyone else in the world has these permissions”.

The third and fourth columns show ownership information: respectively the user and group that own the file.

The seventh and last column shows the file name.

The second column indicates the number of hard links pointing to that file. The fifth column shows the filesize. The sixth column shows the date and time the file was last modified. But these columns are not relevant for the current topic.

What about Directories?

If you try to query information about a directory using `ls -l`, it will show you a listing of the directory's contents instead:

```
$ ls -l Another_Directory/  
total 0  
-rw-r--r-- 1 carol carol 0 Dec 10 17:59 another_file.txt
```

To avoid this and query information about the directory itself, add the `-d` parameter to `ls`:

```
$ ls -l -d Another_Directory/  
drwxrwxr-x 2 carol carol 4096 Dec 10 17:59 Another_Directory/
```

Viewing Hidden Files

The directory listing we have retrieved using `ls -l` before is incomplete:

```
$ ls -l  
total 544  
drwxrwxr-x 2 carol carol 4096 Dec 10 17:59 Another_Directory  
-rw----- 1 carol carol 539663 Dec 10 10:43 picture.jpg  
-rw-rw-r-- 1 carol carol 1881 Dec 10 15:57 text.txt
```

There are three other files in that directory, but they are hidden. On Linux, files whose name starts with a period (.) are automatically hidden. To see them we need to add the `-a` parameter to `ls`:

```
$ ls -l -a  
total 544  
drwxrwxr-x 3 carol carol 4096 Dec 10 16:01 .  
drwxrwxr-x 4 carol carol 4096 Dec 10 15:56 ..  
drwxrwxr-x 2 carol carol 4096 Dec 10 17:59 Another_Directory  
-rw----- 1 carol carol 539663 Dec 10 10:43 picture.jpg  
-rw-rw-r-- 1 carol carol 1881 Dec 10 15:57 text.txt  
-rw-r--r-- 1 carol carol 0 Dec 10 16:01 .thisIsHidden
```

The file `.thisIsHidden` is simply hidden because its name starts with `..`

The directories `.` and `..` however are special. `.` is a pointer to the current directory. And `..` is a pointer to the parent directory, the one which contains the current one. In Linux, every directory contains at least these two directories.

Tip

You can combine multiple parameters for `ls` (and many other Linux commands). `ls -l -a` can, for example, be written as `ls -la`.

Understanding Filetypes

We have already mentioned that the first letter in each output of `ls -l` describes the type of the file. The three most common file types are:

- (normal file)

A file can contain data of any kind and help to manage this data. Files can be modified, moved, copied and deleted.

d (directory)

A directory contains other files or directories and helps to organize the file system. Technically, directories are a special kind of file.

l (symbolic link)

This “file” is a pointer to another file or directory elsewhere in the filesystem.

In addition to these, there are three other file types that you should at least know about, but are out of scope for this lesson:

b (block device)

This file stands for a virtual or physical device, usually disks or other kinds of storage devices, such as the first hard disk which might be represented by `/dev/sda`.

c (character device)

This file stands for a virtual or physical device. Terminals (like the main terminal on `/dev/ttyS0`) and serial ports are common examples of character devices.

s (socket)

Sockets serve as “conduits” passing information between two programs.

Warning

Do not alter any of the permissions on block devices, character devices or sockets, unless you know what you are doing. This may prevent your system from working!

Understanding Permissions

In the output of `ls -l` the file permissions are shown right after the filetype, as three groups of three characters each, in the order r, w and x. Here is what they mean. Keep in mind that a dash - represents the lack of a permission.

Permissions on Files

r

Stands for read and has an octal value of 4 (do not worry, we will discuss octals shortly). This means permission to open a file and read its contents.

w

Stands for write and has an octal value of 2. This means permission to edit or delete a file.

x

Stands for execute and has an octal value of 1. This means that the file can be run as an executable or script.

So, for example, a file with permissions `rw-` can be read and written to, but cannot be executed.

Permissions on Directories

r

Stands for read and has an octal value of 4. This means permission to read the directory's contents, like filenames. But it does not imply permission to read the files themselves.

w

Stands for write and has an octal value of 2. This means permission to create or delete files in a directory.

Keep in mind that you cannot make these changes with write permissions alone, but also need execute permission (x) to change to the directory.

x

Stands for execute and has an octal value of 1. This means permission to enter a directory, but not to list its files (for that `r` is needed).

The last bit about directories may sound a bit confusing. Let us imagine, for example, that you have a directory named `Another_Directory`, with the following permissions:

```
$ ls -ld Another_Directory/
d--x--x--x 2 carol carol 4,0K Dec 20 18:46 Another_Directory
```

Also imagine that inside this directory you have a shell script called `hello.sh`:

```
-rwxr-xr-x 1 carol carol 33 Dec 20 18:46 hello.sh
```

If you are the user `carol` and try to list the contents of `Another_Directory`, you will get an error message, as your user lacks read permission for that directory:

```
$ ls -l Another_Directory/
ls: cannot open directory 'Another_Directory/': Permission denied
```

However, the user `carol` does have execute permissions, which means that she can enter the directory. Therefore, the user `carol` can access files inside the directory, as long as she has the correct permissions for the respective file. Let us assume the user has full permissions (`rwX`) for the script `hello.sh`. Then she can run the script, even though she cannot read the contents of the directory containing it if she knows the complete filename:

```
$ sh Another_Directory/hello.sh
Hello LPI World!
```

As we said before, permissions are specified in sequence: first for the owner of the file, then for the owning group, and then for other users. Whenever someone tries to perform an action on the file, the permissions are checked in the same fashion.

First the system checks if the current user owns the file, and if this is true it applies the first set of permissions only. Otherwise, it checks if the current user belongs to the group owning the file. In that case, it applies the second set of permissions only. In any other case, the system will apply the third set of permissions.

This means that if the current user is the owner of the file, only the owner permissions are effective, even if the group or other permissions are more permissive than the owner's permissions.

Modifying File Permissions

The command `chmod` is used to modify the permissions for a file, and takes at least two parameters: the first one describes which permissions to change, and the second one points to the file or directory where the change will be made. Keep in mind that only the owner of the file, or the system administrator (root) can change the permissions on a file.

The permissions to change can be described in two different ways, or “modes”.

The first one, called symbolic mode offers fine grained control, allowing you to add or revoke a single permission without modifying others on the set. The other mode, called octal mode, is easier to remember and quicker to use if you wish to set all permission values at once.

Both modes will lead to the same end result. So, for example, the commands:

```
$ chmod ug+rw-x,o-rwx text.txt  
and
```

```
$ chmod 660 text.txt
```

will produce exactly the same output, a file with the permissions set:

```
-rw-rw---- 1 carol carol 765 Dec 20 21:25 text.txt
```

Now, let us see how each mode works.

Symbolic Mode

When describing which permissions to change in symbolic mode the first character(s) indicate(s) whose permissions you will alter: the ones for the user (u), for the group (g), for others (o) and/or for everyone (a).

Then you need to tell the command what to do: you can grant a permission (+), revoke a permission (-), or set it to a specific value (=).

Lastly, you specify which permission you wish to act on: read (r), write (w), or execute (x).

For example, imagine we have a file called `text.txt` with the following permission set:

```
$ ls -l text.txt  
-rw-r--r-- 1 carol carol 765 Dec 20 21:25 text.txt
```

If you wish to grant write permissions to members of the group owning the file, you would use the `g+w` parameter. It is easier if you think about it this way: “For the group (g), grant (+) write permissions (w)”. So, the command would be:

```
$ chmod g+w text.txt
```

Let us check the result with `ls`:

```
$ ls -l text.txt  
-rw-rw-r-- 1 carol carol 765 Dec 20 21:25 text.txt
```

Do you wish to remove read permissions for the owner of the same file? Think about it as: “For the user (u), revoke (-) read permissions (r)”. So the parameter is `u-r`, like so:

```
$ chmod u-r text.txt  
$ ls -l text.txt  
--w-rw-r-- 1 carol carol 765 Dec 20 21:25 text.txt
```

What if we want to set the permissions exactly as `rw-` for everyone? Then think of it as: “For all (a), set exactly (=) read (r), write (w), and no execute (-)”. So:

```
$ chmod a=rw- text.txt
$ ls -l text.txt
-rw-rw-rw- 1 carol carol 765 Dec 20 21:25 text.txt
```

Of course, it is possible to modify multiple permissions at the same time. In this case, separate them with a comma (,):

```
$ chmod u+rw,g-x text.txt
$ ls -lh text.txt
-rwxrw-rw- 1 carol carol 765 Dec 20 21:25 text.txt
```

The example above can be read as: “For the user (u), grant (+) read, write and execute (rwx) permissions, for the group (g), revoke (-) execute permissions (x)”.

When run on a directory, `chmod` modifies only the directory’s permissions. `chmod` also has a recursive mode, which is useful for when you want to change the permissions for “all files inside a directory and its subdirectories”. To use this, add the parameter `-R` after the command name, before the permissions to change:

```
$ chmod -R u+rwx Another_Directory/
```

This command can be read as: “Recursively (-R), for the user (u), grant (+) read, write and execute (rwx) permissions”.

Warning

Be careful and think twice before using the `-R` switch, as it is easy to change permissions on files and directories which you do not want to change, especially on directories with a large number of files and subdirectories.

Octal Mode

In octal mode, the permissions are specified in a different way: as a three-digit value on octal notation, a base-8 numeric system.

Each permission has a corresponding value, and they are specified in the following order: first comes read (r), which is 4, then write (w), which is 2 and last is execute (x), represented by 1. If there is no permission, use the value zero (0). So, a permission of `rwx` would be 7 (4+2+1) and `r-x` would be 5 (4+0+1).

The first of the three digits on the permission set represents the permissions for the user (u), the second for the group (g) and the third for others (o). If we wanted to set the permissions for a file to `rw-rw----`, the octal value would be 660:

```
$ chmod 660 text.txt
$ ls -l text.txt
-rw-rw---- 1 carol carol 765 Dec 20 21:25 text.txt
```

Besides this, the syntax in octal mode is the same as in symbolic mode, the first parameter represents the permissions you wish to change, and the second parameter points to the file or directory where the change will be made.

Tip

If a permission value is odd, the file surely is executable!

Which syntax should you use? The octal mode is recommended if you want to change the permissions to a specific value, for example 640 (`rw- r-- ---`).

The symbolic mode is more useful if you want to flip just a specific value, regardless of the current permissions for the file. For example, you can add execute permissions for the user using just `chmod u+x script.sh` without regard to, or even touching, the current permissions for the group and others.

Modifying File Ownership

The command `chown` is used to modify the ownership of a file or directory. The syntax is quite simple:

```
chown USERNAME:GROUPNAME FILENAME
```

For example, let us check a file called `text.txt`:

```
$ ls -l text.txt
```

```
-rw-rw---- 1 carol carol 1881 Dec 10 15:57 text.txt
```

The user who owns the file is `carol`, and the group is also `carol`. Now, we will change the group owning the file to some other group, like `students`:

```
$ chown carol:students text.txt
```

```
$ ls -l text.txt
```

```
-rw-rw---- 1 carol students 1881 Dec 10 15:57 text.txt
```

Keep in mind that the user who owns a file does not need to belong to the group who owns a file. In the example above, the user `carol` does not need to be a member of the `students` group.

The user or group permission set can be omitted if you do not wish to change them. So, to change just the group owning a file you would use `chown :students text.txt`. To change just the user, the command would be `chown carol: text.txt` or just `chown carol text.txt`. Alternatively, you could use the command `chgrp students text.txt`.

Unless you are the system administrator (`root`), you cannot change ownership of a file to another user or group you do not belong to. If you try to do this, you will get the error message `Operation not permitted`.

Querying Groups

Before changing the ownership of a file, it might be useful to know which groups exist on the system, which users are members of a group and to which groups a user belongs.

To see which groups exist on your system, type `getent group`. The output will be similar to this one (the output has been abbreviated):

```
$ getent group
```

```
root:x:0:
```

```
daemon:x:1:
```

```
bin:x:2:
```

```
sys:x:3:
```

```
adm:x:4:syslog,rigues
```

```
tty:x:5:rigues
```

```
disk:x:6:
```

```
lp:x:7:
```

```
mail:x:8:
```



```
news:x:9:
```

```
uucp:x:10:rigues
```

If you want to know to which groups a user belongs, add the username as a parameter to groups:

```
$ groups carol
```

```
carol : carol students cdrom sudo dip plugdev lpadmin sambashare
```

To do the reverse (see which users belong to a group) use groupmems. The parameter -g specifies the group, and -l will list all of its members:

```
# groupmems -g cdrom -l
```

```
carol
```

Tip

groupmems can only be run as root, the system administrator. If you are not currently logged in as root, add sudo before the command.

Special Permissions

Besides the read, write and execute permissions for user, group and others, each file can have three other special permissions which can alter the way a directory works or how a program runs. They can be specified either in symbolic or octal mode, and are as follows:

Sticky Bit

The sticky bit, also called the restricted deletion flag, has the octal value 1 and in symbolic mode is represented by a t within the other's permissions. This applies only to directories, and has no effect on normal files. On Linux it prevents users from removing or renaming a file in a directory unless they own that file or directory.

Directories with the sticky bit set show a t replacing the x on the permissions for others on the output of ls -l:

```
$ ls -ld Sample_Directory/
```

```
drwxr-xr-t 2 carol carol 4096 Dec 20 18:46 Sample_Directory/
```

In octal mode, the special permissions are specified using a 4-digit notation, with the first digit representing the special permission to act upon. For example, to set the sticky bit (value 1) for the directory Another_Directory in octal mode, with permissions 755, the command would be:

```
$ chmod 1755 Another_Directory
```

```
$ ls -ld Another_Directory
```

```
drwxr-xr-t 2 carol carol 4,0K Dec 20 18:46 Another_Directory
```

Set GID

Set GID, also known as SGID or Set Group ID bit, has the octal value 2 and in symbolic mode is represented by an s on the group permissions. This can be applied to executable files or directories. On files, it will make the process run with the privileges of the group who owns the file. When applied to directories, it will make every file or directory created under it inherit the group from the parent directory.

Files and directories with SGID bit show an s replacing the x on the permissions for the group on the output of ls -l:

```
$ ls -l test.sh
-rwxr-sr-x 1 carol root 33 Dec 11 10:36 test.sh
```

To add SGID permissions to a file in symbolic mode, the command would be:

```
$ chmod g+s test.sh
$ ls -l test.sh
-rwxr-sr-x 1 carol root 33 Dec 11 10:36 test.sh
```

The following example will help you better understand the effects of SGID on a directory. Suppose we have a directory called Sample_Directory, owned by the user carol and the group users, with the following permission structure:

```
$ ls -ldh Sample_Directory/
drwxr-xr-x 2 carol users 4,0K Jan 18 17:06 Sample_Directory/
```

Now, let us change to this directory and, using the command touch, create an empty file inside of it. The result would be:

```
$ cd Sample_Directory/
$ touch newfile
$ ls -lh newfile
-rw-r--r-- 1 carol carol 0 Jan 18 17:11 newfile
```

As we can see, the file is owned by the user carol and group carol. But, if the directory had the SGID permission set, the result would be different. First, let us add the SGID bit to the Sample_Directory and check the results:

```
$ sudo chmod g+s Sample_Directory/
$ ls -ldh Sample_Directory/
drwxr-sr-x 2 carol users 4,0K Jan 18 17:17 Sample_Directory/
```

The s on the group permissions indicates that the SGID bit is set. Now, we will change to this directory and, again, create an empty file with the touch command:

```
$ cd Sample_Directory/
$ touch emptyfile
$ ls -lh emptyfile
-rw-r--r-- 1 carol users 0 Jan 18 17:20 emptyfile
```

The group who owns the file is users. This is because the SGID bit made the file inherit the group owner of its parent directory, which is users.

Set UID

SUID, also known as Set User ID, has the octal value 4 and is represented by an s on the user permissions in symbolic mode. It only applies to files and has no effect on directories. Its behavior is similar to the SGID bit, but the process will run with the privileges of the user who owns the file. Files with the SUID bit show a s replacing the x on the permissions for the user on the output of ls -l:

```
$ ls -ld test.sh
-rwsr-xr-x 1 carol carol 33 Dec 11 10:36 test.sh
```

You can combine multiple special permissions on one parameter. So, to set SGID (value 2) and SUID (value 4) in octal mode for the script test.sh with permissions 755, you would type:

```
$ chmod 6755 test.sh
And the result would be:
```

```
$ ls -lh test.sh
-rwsr-sr-x 1 carol carol 66 Jan 18 17:29 test.sh
```

Tip

If your terminal supports color, and these days most of them do, you can quickly see if these special permissions are set by glancing at the output of `ls -l`. For the sticky bit, the directory name might be shown in a black font with blue background. The same applies for files with the SGID (yellow background) and SUID (red background) bits. Colors may be different depending on which Linux distribution and terminal settings you use.

Create and change hard and symbolic links

Introduction

On Linux some files get a special treatment either because of the place they are stored in, such as temporary files, or the way they interact with the filesystem, like links. In this lesson you will learn what links are and how to manage them.

Understanding Links

As already mentioned, on Linux everything is treated as a file. But there is a special kind of file, called a link, and there are two types of links on a Linux system:

Symbolic links

Also called soft links, they point to the path of another file. If you delete the file the link points to (called target) the link will still exist, but it “stops working”, as it now points to “nothing”.

Hard links

Think of a hard link as a second name for the original file. They are not duplicates, but instead are an additional entry in the filesystem pointing to the same place (inode) on the disk.

Tip

An inode is a data structure that stores attributes for an object (like a file or directory) on a filesystem. Among those attributes are permissions, ownership and on which blocks of the disk the data for the object is stored. Think of it as an entry on an index, hence the name, which comes from “index node”.

Working with Hard Links

Creating Hard Links

The command to create a hard link on Linux is `ln`. The basic syntax is:

```
$ ln TARGET LINK_NAME
```

The TARGET must exist already (this is the file the link will point to), and if the target is not on the current directory, or if you want to create the link elsewhere, you must specify the full path to it. For example, the command:

```
$ ln target.txt /home/carol/Documents/hardlink
```

will create a file named `hardlink` on the directory `/home/carol/Documents/`, linked to the file `target.txt` on the current directory.

If you leave out the last parameter (LINK_NAME), a link with the same name as the target will be created in the current directory.

Managing Hard Links

Hard links are entries in the filesystem which have different names but point to the same data on disk. All such names are equal and can be used to refer to a file. If you change the contents of one of the names, the contents of all other names pointing to that file change since all these names point to the very same data. If you delete one of the names, the other names will still work.

This happens because when you “delete” a file the data is not actually erased from the disk. The system simply deletes the entry on the filesystem table pointing to the inode corresponding to the data on the disk. But if you have a second entry pointing to the same inode, you can still get to the data. Think of it as two roads converging on the same point. Even if you block or redirect one of the roads, you can still reach the destination using the other.

You can check this by using the -i parameter of ls. Consider the following contents of a directory:

```
$ ls -li
total 224
3806696 -r--r--r-- 2 carol carol 111702 Jun  7 10:13 hardlink
3806696 -r--r--r-- 2 carol carol 111702 Jun  7 10:13 target.txt
```

The number before the permissions is the inode number. See that both the file hardlink and the file target.txt have the same number (3806696)? This is because one is a hard link of the other.

But which one is the original and which one is the link? You cannot really tell, as for all practical purposes they are the same.

Note that every hard link pointing to a file increases the link count of the file. This is the number right after the permissions on the output of ls -l. By default, every file has a link count of 1 (directories have a count of 2), and every hard link to it increases the count by one. So, that is the reason for the link count of 2 on the files in the listing above.

In contrast to symbolic links, you can only create hard links to files, and both the link and target must reside in the same filesystem.

Moving and Removing Hard Links

Since hard links are treated as regular files, they can be deleted with rm and renamed or moved around the filesystem with mv. And since a hard link points to the same inode of the target, it can be moved around freely, without fear of “breaking” the link.

Symbolic Links

Creating Symbolic Links

The command used to create a symbolic link is also ln, but with the -s parameter added. Like so:

```
$ ln -s target.txt /home/carol/Documents/softlink
```

This will create a file named softlink in the directory /home/carol/Documents/, pointing to the file target.txt on the current directory.

As with hard links, you can omit the link name to create a link with the same name as the target in the current directory.

Managing Symbolic Links

Symbolic links point to another path in the filesystem. You can create soft links to files and

directories, even on different partitions. It is pretty easy to spot a symbolic link with the output of the ls command:

```
$ ls -lh
total 112K
-rw-r--r-- 1 carol carol 110K Jun  7 10:13 target.txt
lrwxrwxrwx 1 carol carol  12 Jun  7 10:14 softlink -> target.txt
```

In the example above, the first character on the permissions for the file softlink is l, indicating a symbolic link. Furthermore, just after the filename you see the name of the target the link points to, the file target.txt.

Note that on file and directory listings, soft links themselves always show the permissions rwx for the user, the group and others, but in practice the access permissions for them are the same as those for the target.

Moving and Removing Symbolic Links

Like hard links, symbolic links can be removed using rm and moved around or renamed using mv. However, special care should be taken when creating them, to avoid “breaking” the link if it is moved from its original location.

When creating symbolic links you should be aware that unless a path is fully specified the location of the target is interpreted as relative to the location of the link. This may create problems if the link, or the file it points to, is moved.

This is easier to understand with an example. Say you have a file named original.txt in the current directory, and you want to create a symbolic link to it called softlink. You could use:

```
$ ln -s original.txt softlink
```

And apparently all would be well. Let us check with ls:

```
$ ls -lh
total 112K
-r--r--r-- 1 carol carol 110K Jun  7 10:13 original.txt
lrwxrwxrwx 1 carol carol  12 Jun  7 19:23 softlink -> original.txt
```

See how the link is constructed: softlink points to (→) original.txt. However, let’s see what happens if you move the link to the previous directory and try to display its contents using the command less:

```
$ mv softlink ../
$ less ../softlink
../softlink: No such file or directory
```

Since the path to original.txt was not specified, the system assumes that it is on the same directory as the link. When this is no longer true, the link stops working.

The way to prevent this is to always specify the full path to the target when creating the link:

```
$ ln -s /home/carol/Documents/original.txt softlink
```

This way, no matter where you move the link to it will still work, because it points to the absolute location of the target. Check with ls:

```
$ ls -lh
total 112K
lrwxrwxrwx 1 carol carol  40 Jun  7 19:34 softlink -> /home/carol/Documents/original.txt
```

Find system files and place files in the correct

location

Introduction

Linux distributions come in all shapes and sizes, but one thing that almost all of them share is that they follow the Filesystem Hierarchy Standard (FHS), which defines a “standard layout” for the filesystem, making interoperation and system administration much easier. In this lesson, you will learn more about this standard, and how to find files on a Linux system.

The Filesystem Hierarchy Standard

The Filesystem Hierarchy Standard (FHS) is an effort by the Linux Foundation to standardize the directory structure and directory contents on Linux systems. Compliance with the standard is not mandatory, but most distributions follow it.

Note

Those interested in the details of filesystem organization can read the FHS 3.0 specification, available in multiple formats at: <http://refspecs.linuxfoundation.org/fhs.shtml>

According to the standard, the basic directory structure is as follows:

/

This is the root directory, the topmost directory in the hierarchy. Every other directory is located inside it. A filesystem is often compared to a “tree”, so this would be the “trunk” to which all branches are connected.

/bin

Essential binaries, available to all users.

/boot

Files needed by the boot process, including the Initial RAM Disk (initrd) and the Linux kernel itself.

/dev

Device files. These can be either physical devices connected to the system (for example, /dev/sda would be the first SCSI or SATA disk) or virtual devices provided by the kernel.

/etc

Host-specific configuration files. Programs may create subdirectories under /etc to store multiple configuration files if needed.

/home

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. The exceptions are the superuser (root), which has a separate directory (/root) and some system users.

/lib

Shared libraries needed to boot the operating system and to run the binaries under /bin and /sbin.

/media

User-mountable removable media, like flash drives, CD and DVD-ROM readers, floppy disks, memory cards and external disks are mounted under here.

/mnt

Mount point for temporarily mounted filesystems.

/opt

Application software packages.

/root

Home directory for the superuser (root).

/run

Run-time variable data.

/sbin

System binaries.

/srv

Data served by the system. For example, the pages served by a web server could be stored under /srv/www.

/tmp

Temporary files.

/usr

Read-only user data, including data needed by some secondary utilities and applications.

/proc

Virtual filesystem containing data related to running processes.

/var

Variable data written during system operation, including print queue, log data, mailboxes, temporary files, browser cache, etc.

Keep in mind that some of those directories, like /etc, /usr and /var, contain a whole hierarchy of subdirectories under them.

Temporary Files

Temporary files are files used by programs to store data that are only needed for a short time. This can be the data of running processes, crash logs, scratch files from an autosave, intermediary files used during a file conversion, cache files and such.

Location of Temporary Files

Version 3.0 of the Filesystem Hierarchy Standard (FHS) defines standard locations for temporary files on Linux systems. Each location has a different purpose and behavior, and it is recommended that developers follow the conventions set by the FHS when writing temporary data to disk.

/tmp

According to the FHS, programs should not assume that files written here will be preserved between invocations of a program. The recommendation is that this directory be cleared (all files erased) during system boot-up, although this is not mandatory.

/var/tmp

Another location for temporary files, but this one should not be cleared during the system boot-up. Files stored here will usually persist between reboots.

/run

This directory contains run-time variable data used by running processes, such as process identifier files (.pid). Programs that need more than one run-time file may create subdirectories here. This location must be cleared during system boot-up. The purpose of this directory was once served by /var/run, and on some systems /var/run may be a symbolic link to /run.

Note that there is nothing which prevents a program from creating temporary files elsewhere on the system, but it is good practice to respect the conventions set by the FHS.

Finding Files

To search for files on a Linux system, you can use the find command. This is a very powerful tool, full of parameters that can suit its behaviour and modify output exactly to your needs.

To start, find needs two arguments: a starting point and what to look for. For example, to search for all files in the current directory (and subdirectories) whose name ends in .jpg you can use:

```
$ find . -name '*.jpg'
./pixel_3a_seethrough_1.jpg
./Mate3.jpg
./Expert.jpg
./Pentaro.jpg
./Mate1.jpg
./Mate2.jpg
./Sala.jpg
./Hotbit.jpg
```

This will match any file whose last four characters of the name are .jpg, no matter what comes before it, as * is a wildcard for “anything”. However, see what happens if another * is added at the end of the pattern:

```
$ find . -name '*.jpg*'
./pixel_3a_seethrough_1.jpg
./Pentaro.jpg.zip
./Mate3.jpg
./Expert.jpg
./Pentaro.jpg
./Mate1.jpg
./Mate2.jpg
./Sala.jpg
./Hotbit.jpg
```

The file Pentaro.jpg.zip (highlighted above) was not included in the previous listing, because even if it contains .jpg on its name, it did not match the pattern as there were extra characters after it. The new pattern means “anything .jpg anything”, so it matches.

Tip

Keep in mind that the -name parameter is case sensitive. If you wish to do a case-insensitive search, use -iname.

The *.jpg expression must be placed inside single quotes, to avoid the shell from interpreting the pattern itself. Try without the quotes and see what happens.

By default, find will begin at the starting point and descend through any subdirectories (and subdirectories of those subdirectories) found. You can restrict this behaviour with the -maxdepth N

parameters, where N is the maximum number of levels.

To search only the current directory, you would use `-maxdepth 1`. Suppose you have the following directory structure:

```
directory
├── clients.txt
├── partners.txt -> clients.txt
├── somedir
│   ├── anotherdir
│   └── clients.txt
```

To search inside `somedir`, you would need to use `-maxdepth 2` (the current directory +1 level down). To search inside `anotherdir`, `-maxdepth 3` would be needed (the current directory +2 levels down). The parameter `-mindepth N` works in the opposite way searching only in directories at least N levels down.

The `-mount` parameter can be used to avoid find going down inside mounted filesystems. You can also restrict the search to specific filesystem types using the `-fstype` parameter. So `find /mnt -fstype exfat -iname "*report*"` would only search inside exFAT filesystems mounted under `/mnt`.

Searching for Attributes

You can use the parameters below to search for files with specific attributes, like ones that are writable by your user, have a specific set of permissions or are of a certain size:

`-user USERNAME`

Matches files owned by the user `USERNAME`.

`-group GROUPNAME`

Matches files owned by the group `GROUPNAME`.

`-readable`

Matches files that are readable by the current user.

`-writable`

Matches files that are writable by the current user.

`-executable`

Matches files that are executable by the current user. In the case of directories, this will match any directory that the user can enter (x permission).

`-perm NNNN`

This will match any files that have exactly the `NNNN` permission. For example, `-perm 0664` will match any files which the user and group can read and write to and which others can read (or `rw-rw-r--`).

You can add a `-` before `NNNN` to check for files that have at least the permission specified. For example, `-perm -644` would match files that have at least `644` (`rw-r--r--`) permissions. This includes a file with `664` (`rw-rw-r--`) or even `775` (`rw-rwx-r-x`).

`-empty`

Will match empty files and directories.

`-size N`

Will match any files of N size, where N by default is a number of 512-byte blocks. You can add suffixes to N for other units: `Nc` will count the size in bytes, `Nk` in kibibytes (KiB, multiples of 1024

bytes), NM in mebibytes (MiB, multiples of $1024 * 1024$) and NG for gibibytes (GiB, multiples of $1024 * 1024 * 1024$).

Again, you can add the + or - prefixes (here meaning bigger than and smaller than) to search for relative sizes. For example, -size -10M will match any file less than 10 MiB in size.

For example, to search for files under your home directory which contain the case insensitive pattern report in any part of the name, have 0644 permissions, have been accessed 10 days ago and which size is at least 1 Mib, you could use

```
$ find ~ -iname "*report*" -perm 0644 -atime 10 -size +1M
```

Searching by Time

Besides searching for attributes you can also perform searches by time, finding files that were accessed, had their attributes changed or were modified during a specific period of time. The parameters are:

-amin N, -cmin N, -mmin N

This will match files that have been accessed, had attributes changed or were modified (respectively) N minutes ago.

-atime N, -ctime N, -mtime N

This will match files that were accessed, had attributes changed or were modified N*24 hours ago.

For -cmin N and -ctime N, any attribute change will cause a match, including a change in permissions, reading or writing to the file. This makes these parameters especially powerful, since practically any operation involving the file will trigger a match.

The following example would match any file in the current directory which has been modified less than 24 hours ago and is bigger than 100 MiB:

```
$ find . -mtime -1 -size +100M
```

Using locate and updatedb

locate and updatedb are commands that can be used to quickly find a file matching a given pattern on a Linux system. But unlike find, locate will not search the filesystem for the pattern: instead, it looks it up on a database built by running the updatedb command. This gives you very quick results, but they may be imprecise depending on when the database was last updated.

The simplest way to use locate is to just give it a pattern to search for. For example, to find every JPEG image on your system, you would use locate jpg. The list of results can be quite lengthy, but should look like this:

```
$ locate jpg
/home/carol/Downloads/Expert.jpg
/home/carol/Downloads/Hotbit.jpg
/home/carol/Downloads/Mate1.jpg
/home/carol/Downloads/Mate2.jpg
/home/carol/Downloads/Mate3.jpg
```

```
/home/carol/Downloads/Pentaro.jpg
/home/carol/Downloads/Sala.jpg
/home/carol/Downloads/pixel_3a_seethrough_1.jpg
/home/carol/Downloads/jpg_specs.doc
```

When asked for the jpg pattern, locate will show anything that contains this pattern, no matter what comes before or after it. You can see an example of this in the file jpg_specs.doc in the listing above: it contains the pattern, but the extension is not jpg.

Tip

Remember that with locate you are matching patterns, not file extensions.

By default the pattern is case sensitive. This means that files containing .JPG would not be shown since the pattern is in lowercase. To avoid this, pass the -i parameter to locate. Repeating our previous example:

```
$ locate -i .jpg
/home/carol/Downloads/Expert.jpg
/home/carol/Downloads/Hotbit.jpg
/home/carol/Downloads/Mate1.jpg
/home/carol/Downloads/Mate1_old.JPG
/home/carol/Downloads/Mate2.jpg
/home/carol/Downloads/Mate3.jpg
/home/carol/Downloads/Pentaro.jpg
/home/carol/Downloads/Sala.jpg
/home/carol/Downloads/pixel_3a_seethrough_1.jpg
```

Notice that the file Mate1_old.JPG, in bold above, was not present in the previous listing.

You can pass multiple patterns to locate, just separate them with spaces. The example below would do a case-insensitive search for any files matching the zip and jpg patterns:

```
$ locate -i zip jpg
/home/carol/Downloads/Expert.jpg
/home/carol/Downloads/Hotbit.jpg
/home/carol/Downloads/Mate1.jpg
/home/carol/Downloads/Mate1_old.JPG
/home/carol/Downloads/Mate2.jpg
/home/carol/Downloads/Mate3.jpg
/home/carol/Downloads/OPENMSXPIHAT.zip
/home/carol/Downloads/Pentaro.jpg
/home/carol/Downloads/Sala.jpg
/home/carol/Downloads/gbs-control-master.zip
/home/carol/Downloads/lineage-16.0-20190711-MOD-quark.zip
/home/carol/Downloads/pixel_3a_seethrough_1.jpg
/home/carol/Downloads/jpg_specs.doc
```

When using multiple patterns, you can request locate to show only files that match all of them. This is done with the -A option. The following example would show any file matching the .jpg and the .zip patterns:

```
$ locate -A .jpg .zip
/home/carol/Downloads/Pentaro.jpg.zip
```

If you wish to count the number of files that match a given pattern instead of showing their full path you can use the -c option. For example, to count the number of .jpg files on a system:

```
$ locate -c .jpg
1174
```

One problem with locate is that it only shows entries present in the database generated by updatedb (located in /var/lib/mlocate.db). If the database is outdated, the output could show files that have

been deleted since the last time it was updated. One way to avoid this is to add the `-e` parameter, which will make it check to see if the file still exists before showing it on the output.

Of course, this will not solve the problem of files created after the last database update not showing up. For this you will have to update the database with the command `updatedb`. How long this will take will depend on the amount of files of your disk.

Controlling the Behavior of `updatedb`

The behavior of `updatedb` can be controlled by the file `/etc/updatedb.conf`. This is a text file where each line controls one variable. Blank lines are ignored and lines that start with the `#` character are treated as comments.

`PRUNEFS=`

Any filesystem types indicated after this parameter will not be scanned by `updatedb`. The list of types should be separated by spaces, and the types themselves are case-insensitive, so `NFS` and `nfs` are the same.

`PRUNENAMES=`

This is a space-separated list of directory names that should not be scanned by `updatedb`.

`PRUNEPATHS=`

This is a list of path names that should be ignored by `updatedb`. The path names must be separated by spaces and specified in the same way they would be shown by `updatedb` (for example, `/var/spool /media`)

`PRUNE_BIND_MOUNTS=`

This is a simple yes or no variable. If set to yes bind mounts (directories mounted elsewhere with the `mount --bind` command) will be ignored.

Finding Binaries, Manual Pages and Source Code

which is a very useful command that shows the full path to an executable. For example, if you want to locate the executable for `bash`, you could use:

```
$ which bash
/usr/bin/bash
```

If the `-a` option is added the command will show all pathnames that match the executable. Observe the difference:

```
$ which mkfs.ext3
/usr/sbin/mkfs.ext3
```

```
$ which -a mkfs.ext3
/usr/sbin/mkfs.ext3
/sbin/mkfs.ext3
```

Tip

To find which directories are in the `PATH` use the `echo $PATH` command. This will print (echo) the contents of the variable `PATH` (`$PATH`) to your terminal.

`type` is a similar command which will show information about a binary, including where it is located and its type. Just use `type` followed by the command name:

```
$ type locate
```

locate is /usr/bin/locate

The -a parameter works in the same way as in which, showing all pathnames that match the executable. Like so:

```
$ type -a locate
```

```
locate is /usr/bin/locate
```

```
locate is /bin/locate
```

And the -t parameter will show the file type of the command which can be alias, keyword, function, builtin or file. For example:

```
$ type -t locate
```

```
file
```

```
$ type -t ll
```

```
alias
```

```
$ type -t type
```

```
type is a built-in shell command
```

The command whereis is more versatile and besides binaries can also be used to show the location of man pages or even source code for a program (if available in your system). Just type whereis followed by the binary name:

```
$ whereis locate
```

```
locate: /usr/bin/locate /usr/share/man/man1/locate.1.gz
```

The results above include binaries (/usr/bin/locate) and compressed manual pages (/usr/share/man/man1/locate.1.gz).

You can quickly filter the results using commandline switches like -b, which will limit them to only the binaries, -m, which will limit them to only man pages, or -s, which will limit them to only the source code. Repeating the example above, you would get:

```
$ whereis -b locate
```

```
locate: /usr/bin/locate
```

```
$ whereis -m locate
```

```
locate: /usr/share/man/man1/locate.1.gz
```