

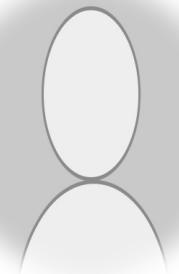
Universidad de San Carlos de Guatemala
Facultad de Ingeniería
Escuela de Ciencias y Sistemas
Sistemas Operativos



Sistemas Operativos

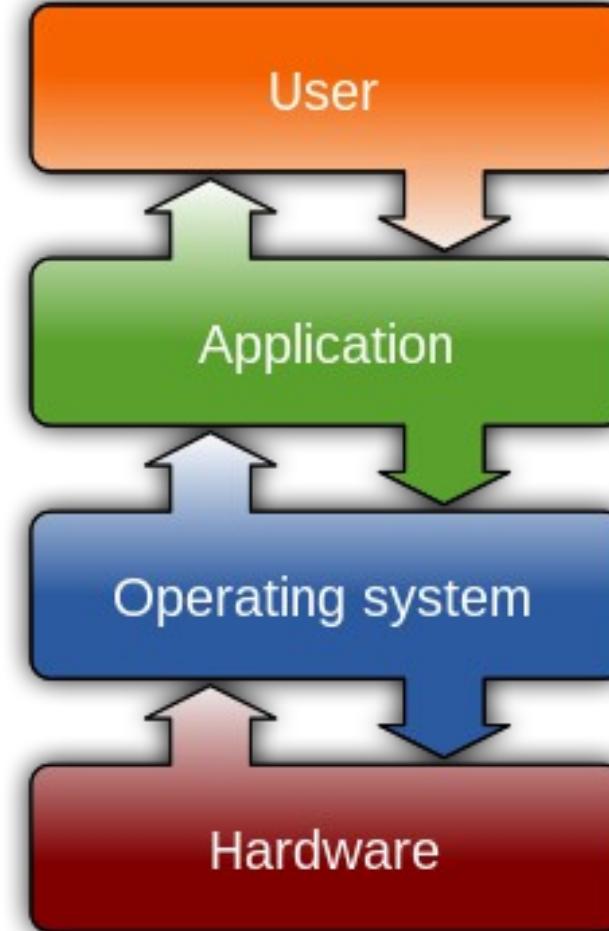
Historia y Conceptos Básicos

Guatemala, julio 2024

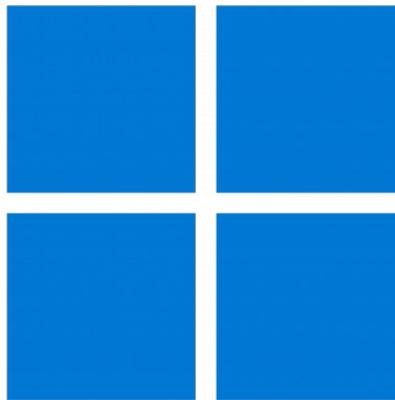


Sistema Operativo

Un sistema operativo (OS) es un software de sistema que administra el hardware de la computadora, los recursos de software y proporciona servicios comunes para los programas de la computadora.



Ejemplos de sistemas operativos

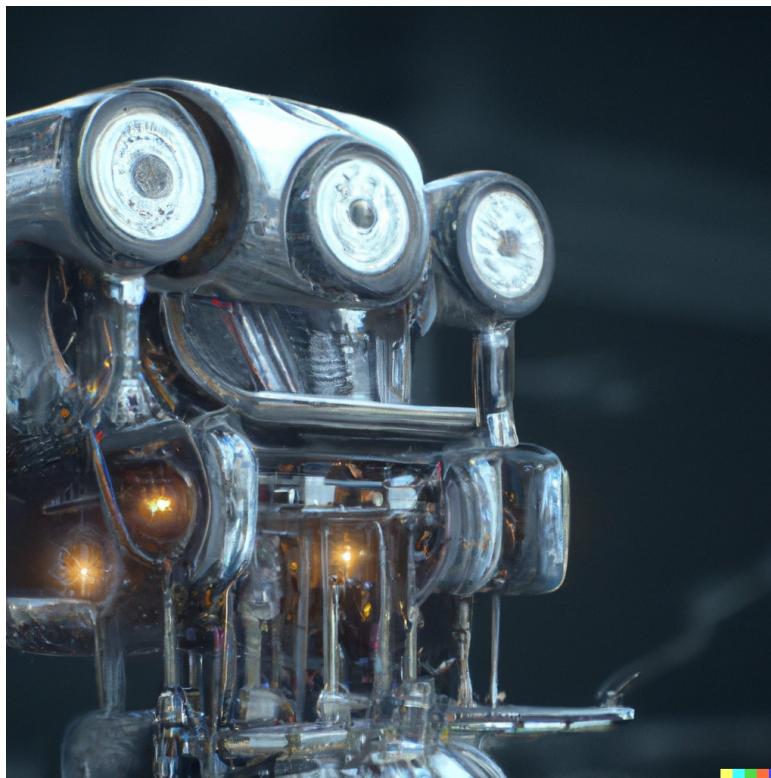


Historia de los sistemas operativos

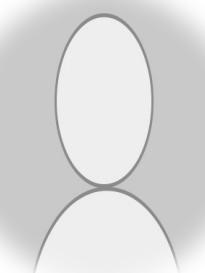
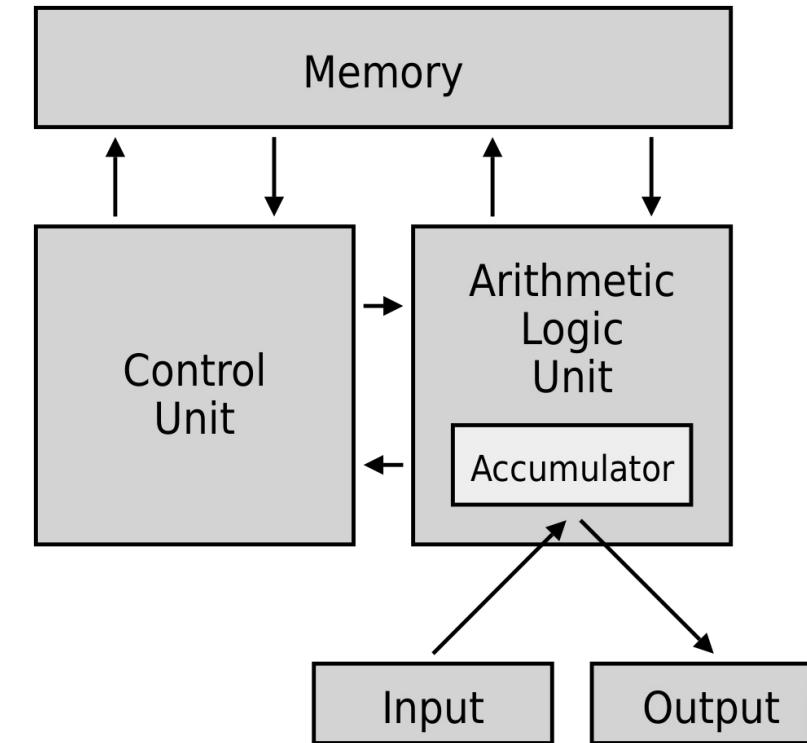


Computadoras de propósito general

Maquina Universal de Turing



Arquitectura Von Neumann



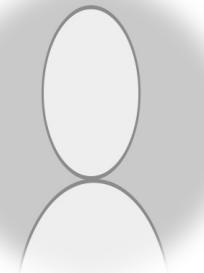
¿Por qué un Sistemas Operativo?

- Administración de recursos
- Ejecución de múltiples programas
- Utilización por múltiples usuarios
- Interface de usuario
- Un sistema de archivos
- Proporcionar servicios de red
- Seguridad
- Centrarnos en la lógica del negocio

¿Es posible que un programa se ejecute sin sistema operativo?

Si

"Technically speaking, an OS is not strictly needed. There are systems out there that do not have an OS. These are usually embedded systems with a tiny footprint: think of an IoT beacon. They simply do not have the resources available to keep anything else around other than one application. For example, with Rust you can use its Core and Standard Library to run any app on [bare metal](#)."



Unix

- Es una familia de sistemas operativos de computadora ***multitarea*** y ***multiusuario*** que derivan del AT&T Unix original, cuyo desarrollo comenzó en 1969 en el centro de investigación Bell Labs por Ken Thompson, Dennis Ritchie y otros.

UNIX®
An Open Group Standard

Conceptos de Unix

- Usar texto plano para guardar data
- Un sistema de archivos jerárquico
- Dispositivos como archivos
- Inter-process Communication como archivo
- Usar muchos programas que hacen las cosas bien
- Pequeños programas que pueden interactuar entre sí por medio de línea de comandos

A todos los conceptos anteriores se les conoce como la "***Filosofía Unix***".



Filosofía Unix



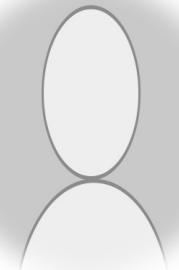
Write programs that do one thing
and do it well.



Write programs to work together.

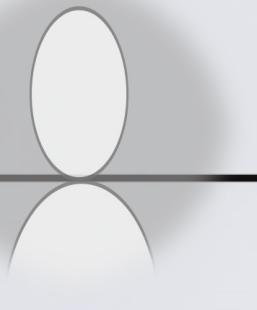
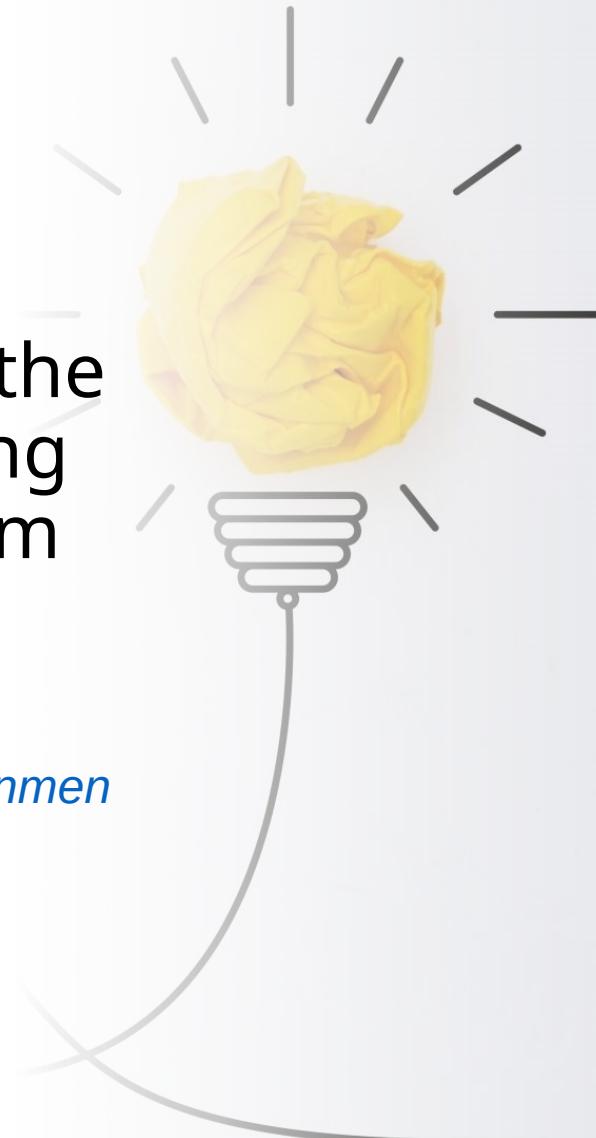


Write programs to handle text
streams, because that is a universal
interface.



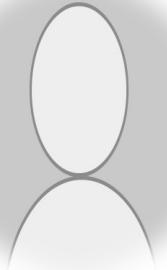
"the idea that the power of a system comes more from the relationships among programs than from the programs themselves"

The Unix Programming Environment



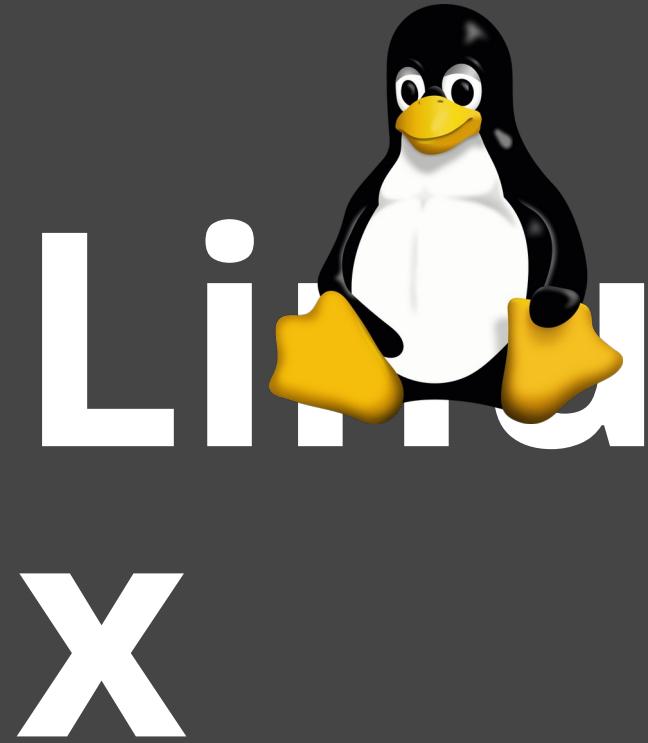
El proyecto GNU

- Proyecto de colaboración de software libre anunciado por Richard Stallman el 27 de septiembre de 1983.
- Brindar a los usuarios libertad y control mediante el desarrollo colaborativo y la publicación de software con todos los derechos de libre acceso, ejecutar, copiar y distribuir, estudiar y modificar, incluso incluyendo el sistema operativo.
- GNU es un acrónimo recursivo que significa "***GNU no es Unix!***".



GNU Hurd

- Según su manifiesto, el objetivo fundacional del proyecto es construir un **sistema operativo** libre y, si es posible, "todas las utilidades que normalmente acompañan a un sistema Unix para que se pueda prescindir de cualquier software que no sea libre".
- Hasta el momento, el proyecto GNU no ha lanzado una versión de GNU/Hurd que sea adecuada para entornos de producción desde el comienzo del proyecto GNU/Hurd hace más de **32 años**.



Anuncio de Linux

Hello everybody out there using minix -

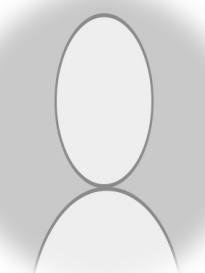
I'm doing a (free) operating system (just a hobby, won't be big and professional like gnu) for 386(486) AT clones. This has been brewing since april, and is starting to get ready. I'd like any feedback on things people like/dislike in minix, as my OS resembles it somewhat (same physical layout of the file-system (due to practical reasons) among other things).

I've currently ported [bash\(1.08\)](#) and [gcc\(1.40\)](#), and things seem to work. This implies that I'll get something practical within a few months, and I'd like to know what features most people would want. Any suggestions are welcome, but I won't promise I'll implement them :-)

Linus (torvalds@kruuna.helsinki.fi)

PS. Yes - it's free of any minix code, and it has a multi-threaded fs. It is NOT portable (uses 386 task switching etc), and it probably never will support anything other than AT-harddisks, as that's all I have :-.

— Linus Torvalds^[17]



Características del Linux Kernel

Multiusuario,
multitarea

Kernel
Monolítico

Portable

Memoria
Virtual

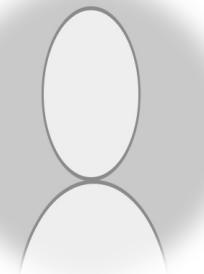
Manejo de
procesos,
archivos y
dispositivos

Controlador de
Arranque

Modularidad

Control de
recursos

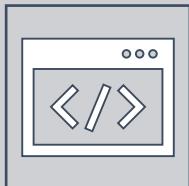
Código Abierto



Linux basado en Unix?



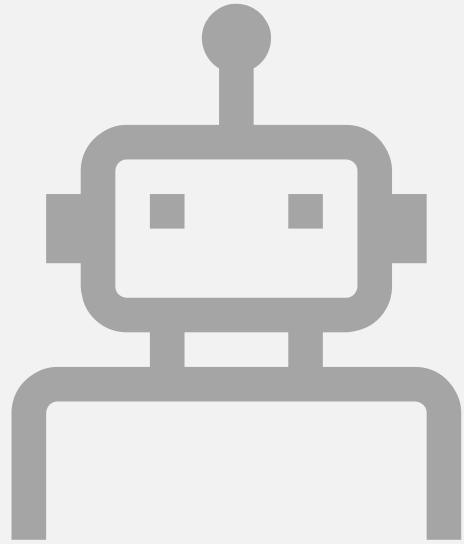
Linux es considerado un sistema operativo basado en Unix porque comparte muchas similitudes en su arquitectura y diseño con Unix.



Al igual que Unix, Linux tiene un diseño modular, un enfoque en la línea de comandos, y un sistema de archivos jerárquico. También comparten similitudes en el uso de sistemas de llamadas al sistema y la forma en que manejan procesos y memoria.



Tipos de Sistema Operativo



Existen varios tipos de sistemas operativos, de las cuales algunos son:

- Único – Múltiple usuario
- Único – Múltiple tarea
- Distribuidos
- En tiempo real
- Embebidos

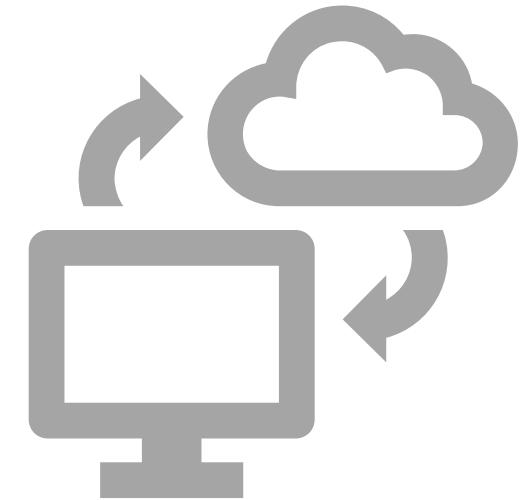
Componentes de Sistema Operativo

- El Kernel
- Sistema de archivos
- Interface de Usuario
- Controladores
- Librerías del sistema
- Gestor de Paquetes
- Servicios de Red
- Servicios de Seguridad
- Herramientas de administración

Sistemas Operativos en la nube

Los sistemas operativos son fundamentales para la infraestructura de la nube, y se utilizan en varias formas, algunos ejemplos son:

- Virtualización
- Contenedores
- Automatización
- Escalabilidad
- Controles de seguridad
- Servicios de Red



Actividad 1

Realizar lo siguiente:

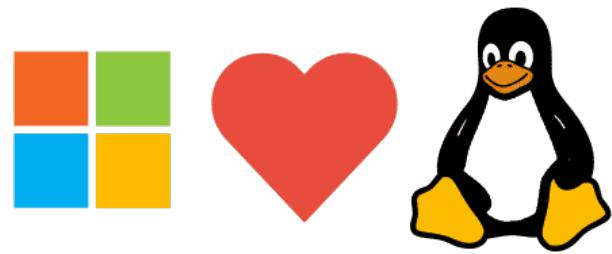
- Preguntar a dos personas que saben sobre los siguientes conceptos: Sistema operativo, archivo e IP.
- Realizar una conclusión en base a los comentarios obtenidos

Fecha de entrega: 21/07/24

Entrega: UEDI. Enviar link a MD file en un repositorio de GitHub

Nombre del repositorio: **so1_actividades_#carnet**

Nombre del folder: **actividad1**



Muchas Gracias

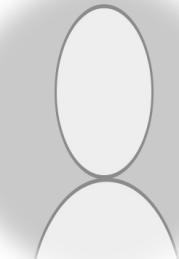
Universidad de San Carlos de Guatemala
Facultad de Ingeniería
Escuela de Ciencias y Sistemas
Sistemas Operativos



Shells and Scripting

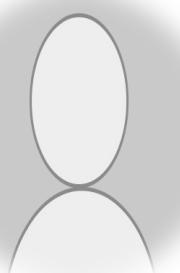
Interactuando con Linux en usando la Terminal

Guatemala, julio 2023



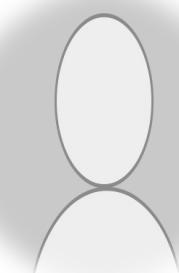
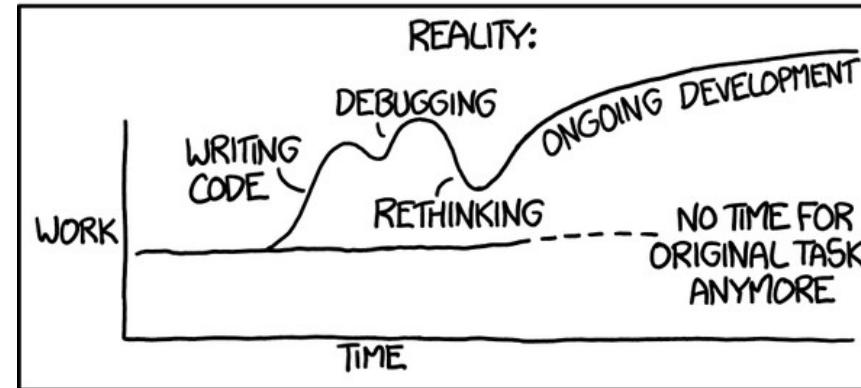
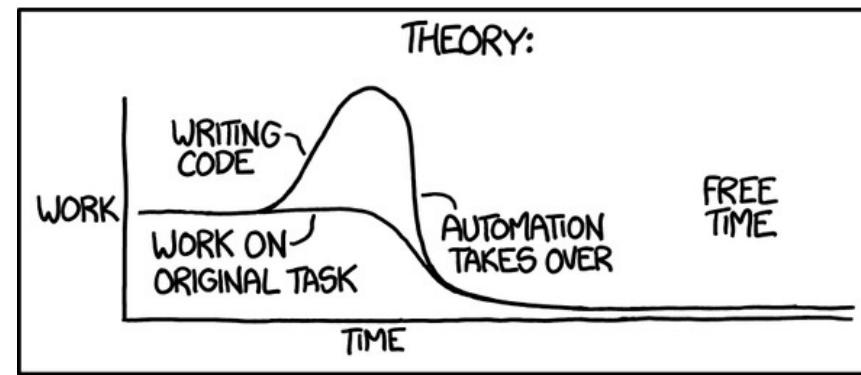
Interactuando con Linux

- Existen dos formas principales para interactuar con Linux, desde una perspectiva de CLI (Command Line Interface).
 - **La forma manual**, usando una terminal. Un humano ejecuta comandos desde una terminal.
 - **La forma automatizada**, donde una serie de comandos son escritos en un archivo (scripts) y ejecutados por la Shell. Comúnmente conocido como *Shell scripting* o solo *scripting*.



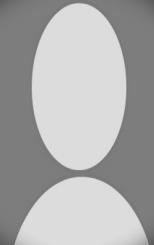
Automatizar ...

"I SPEND A LOT OF TIME ON THIS TASK.
I SHOULD WRITE A PROGRAM AUTOMATING IT!"



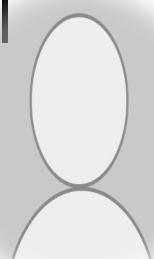
Terminales

- Una terminal es un programa que provee una interface textual
- Soporta leer caracteres del teclado e imprimir texto en la pantalla
- Adicionalmente a caracteres básicos es posible introducir caracteres de escape
- Es posible obtener información de la terminal actual consultando la variable de entorno `TERM` o con el comando `infocmp`
- Hoy en día es común utilizar un emulador de terminal (apps), pero antes era hardware dedicado.



Shells

- Una Shell es un programa que ejecuta dentro de la terminal y sirve como intérprete
- Ofrece streams de entrada y salida (I/O), comandos, variables, maneja la ejecución de comandos y permite la automatización usando scripts.
- Bourne Shell (sh), es el intérprete estándar de comandos. Una [POSIX Shell](#).
- **Bash** (Bourne Again Shell), es la Shell más utilizada y es fue creada para ser un remplazo de *sh*
- La variable de entorno **SHELL** o **\$0** tiene información de la Shell actual

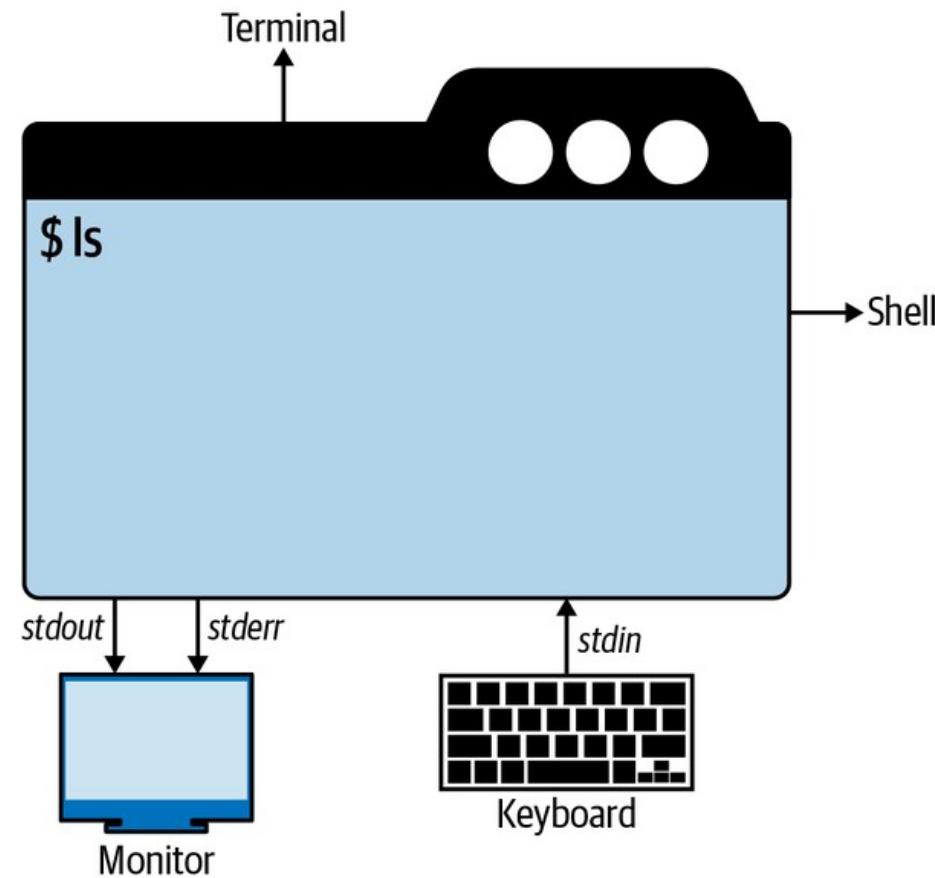


Streams

- Los streams (I/O) son entradas y salidas de datos de nuestra Shell
- Cada proceso en que ejecuta una Shell tiene tres file descriptors (FDs) para salida y entrada
 - stdin (FD 0)
 - stdout (FD 1)
 - stderr (FD 2)
- Cada FD está conectado al teclado y a la pantalla por defecto
- Es posible cambiar el origen y destino (redirección) de los FDs usando pipes



Terminal, Shell and Streams



Redirecciones de Streams

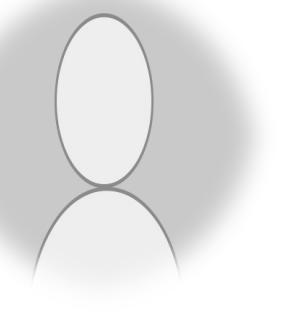
Es posible redirigir un FD usando `$FD>` y `<$FD` donde `$FD` es el descriptor.

`<` para redirigir el stream *stdin*

`>` y `1>` para redirigir el stream *stdout*

`2>` para redirigir el stream *stderr*

`&>` para redirigir *stdout* y *stderr*. Comúnmente a [`/dev/null`](#)



Caracteres especiales en la Shell

Ampersand (&)

- Colocado al final del comando. Ejecuta el comando en background

Backslash (\)

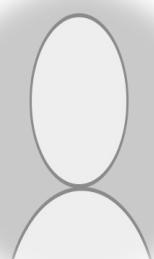
- Usado para continuar el comando en la siguiente línea

Pipe (|)

- Conecta el *stdout* de un proceso con el *stdin* del siguiente proceso. Permite pasar data a otro proceso sin guardarla

Variables en Shells

- Las variables son usadas para guardar valores a los cuales los procesos acceden y las usan para cambiar su comportamiento
- Tipos de variables:
 - Variables de entorno: Configurados en toda la Shell. Es posible listarla con el comando `env`
 - Variables de Shell: Validas en el contexto de ejecución. Es posible listarlas con el comando `set`. Estas no son heredadas a los subprocessos
- Las variables de entorno pueden ser seteadas con `export` y dejar de usarla `unset`
- Es posible acceder a las variables colocando `$` enfrente de la variable



```
$ set MY_VAR=42
$ set | grep MY_VAR
_=MY_VAR=42

$ export MY_GLOBAL_VAR="fun with vars"

$ set | grep 'MY_*
MY_GLOBAL_VAR='fun with vars'
_=MY_VAR=42

$ env | grep 'MY_*
MY_GLOBAL_VAR=fun with vars

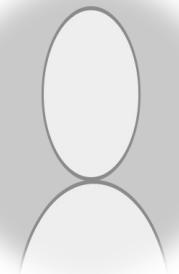
$ bash
$ echo $MY_GLOBAL_VAR
fun with vars

$ set | grep 'MY_*
MY_GLOBAL_VAR='fun with vars'

$ exit
$ unset $MY_VAR
$ set | grep 'MY_*
MY_GLOBAL_VAR='fun with vars'
```

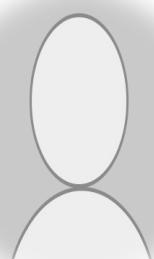
Variables de Shell Comunes

Variable	Semantics
HOME	The path of the home directory of the current user
HOSTNAME	The name of the current host
PATH	Contains a list of directories in which the shell looks for executable programs (binaries or scripts)
PWD	The full path of the working directory
OLDPWD	The full path of the directory before the last cd command
RANDOM	A random integer between 0 and 32767
SHELL	Contains the currently used shell
TERM	The terminal emulator used
UID	Current user unique ID (integer value)
USER	Current user name
_	Last argument to the previous command executed in the foreground
?	Exit status; see " Exit status "
\$	The ID of the current process (integer value)
0	The name of the current process



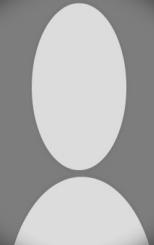
Exit Status

- La Shell comunica la finalización de un comando con algo llamado *exit status*
- Se espera que un comando de Linux retorne su estado cuando este termine. No importa que haya terminado correctamente o incorrectamente
- Las posibles salidas son el 0 a 255. El 0 indica que todo termino correctamente
- Para consultar el exit status del último comando echo \$?



Job Control

- Job Control toma el control de comandos ejecutados en segundo plano (*background*)
- Por defecto cuando se ejecuta un comando esta toma control de la pantalla y el teclado (*foreground*)
- Para lanzar un comando en *background* debes de agregar & al final del comando
- Para lanzar un comando desde *foreground* to *background* presiona Ctrl+Z
- Si es necesario ejecutar un comando incluso si la Shell se cierra es necesario usar el comando *nohup*



```
$ watch -n 5 "ls" &

# List all jobs
$ jobs
Job      Group      CPU      State      Command
1          3021      0%      stopped    watch -n 5 "ls" &

# Pick up a background process to foreground
$ fg

$ nohup watch -n 5 "ls" &
# Close the shell, the process will continue

# In another Shell, check the process
$ ps j -A

# Kill a process
$ kill -9 <process_number>
```

Otras
funciones
de Bash

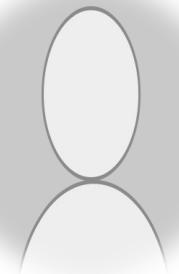
Palabras reservadas

Lista de comandos

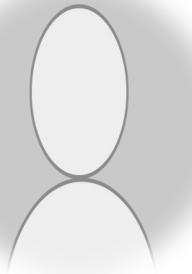
Pipelines

Condicionales

Ciclos

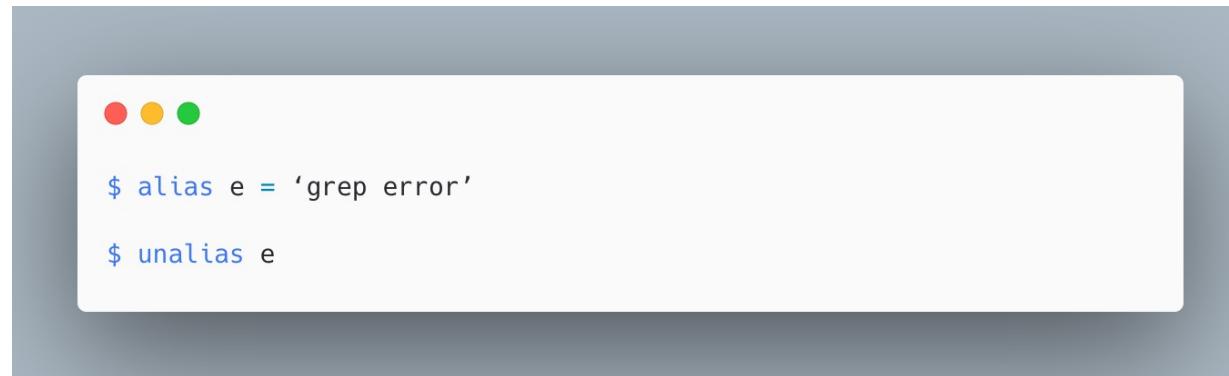


Tareas Comunes



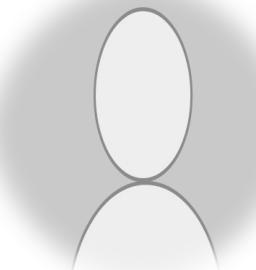
Acortar comandos comunes

- Utilizar el comando alias para acortar comandos más utilizados
 - alias e = 'grep error'
- Utilizar el comando unalias para eliminar un alias
 - unalias e



A screenshot of a macOS terminal window. The window has a dark grey header bar with three colored dots (red, yellow, green) in the top-left corner. The main pane is white and contains the following text:

```
$ alias e = 'grep error'  
$ unalias e
```



Navegación

ACTION	COMMAND	NOTE
Move cursor to start of line	Ctrl+a	-
Move cursor to end of line	Ctrl+e	-
Move cursor forward one character	Ctrl+f	-
Move cursor back one character	Ctrl+b	-
Move cursor forward one word	Alt+f	Works only with left Alt
Move cursor back one word	Alt+b	-
Delete current character	Ctrl+d	-
Delete character left of cursor	Ctrl+h	-
Delete word left of cursor	Ctrl+w	-
Delete everything right of cursor	Ctrl+k	-
Delete everything left of cursor	Ctrl+u	-
Clear screen	Ctrl+l	-
Cancel command	Ctrl+c	-
Undo	Ctrl+_	bash only
Search history	Ctrl+r	Some shells

Manejo de archivos

- Cuando usamos > escribimos a un file (overwrite)
- Cuando usamos >> escribimos al final de un archivo (append)
- El comando sed ayuda a hacer sustituciones
- El comando diff muestra las diferencias entre archivos

```
$ echo "First line" > /tmp/something
$ cat /tmp/something
First line

$ echo "Second line" >> /tmp/something && \
  cat /tmp/something
First line
Second line

$ sed 's/line/LINE/' /tmp/something
First LINE
Second LINE

# we can use vi or nano as text editor
$ cat << 'EOF' > /tmp/another
First line
Second line
Third line
EOF

$ diff -y /tmp/something /tmp/another
First line
First line
Second line
```

Ver logs

- Los líneas logs tienen a tener muchas líneas. Comandos como **less** o **bat** nos ayudan paginar estos archivos y navegar de una mejor manera
- El comando **tail** se utiliza para obtener cambios de un archivo
- El comando head para obtener las primeas líneas de un archivo

```
$ for i in {1..100} ; do echo $i >> /tmp/longfile ; done  
$ head -5 /tmp/longfile
```

Fechas

- Utilizar el comando date para obtener la fecha actual
- Es posible generar múltiples formatos de fecha
- Utiliza ***Unix time stamp*** que es el número de segundos transcurridos desde 1970-01-01T00:00:00Z. El tiempo de UNIX trata todos los días con una duración exacta de 86.400 segundos.
- Es común generar nombres de archivos utilizando date -s



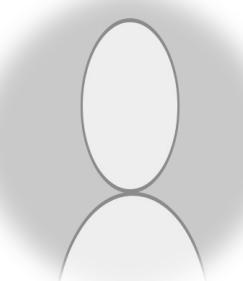
A screenshot of a terminal window with a dark background and light-colored text. It shows three lines of command-line input:

```
$ date +%s  
1629582883  
  
$ date -d @1629742883 '+%m/%d/%Y:%H:%M:%S'
```

Shells amigables para el usuario

Bash es la Shell más utilizada, pero no es necesariamente la más amigable. Fue desarrollada en 1980's. Las siguientes pueden ser unas alternativas con más poderes.

- [Fish Shell](#)
- [Zshell \(zsh\)](#)
- [Powershell](#)
- [Oil Shell](#)
- [Murex](#)



Scripting



Scripting

- Son archivos de texto que se utilizan para automatizar tareas. Ellos contienen una lista de instrucciones que serán ejecutadas
- La ejecución de los scripts puede ser manual, pero es común programar la ejecución usando cronjobs o algún trigger externo
- Por lo general los scripts están escritos en bash debido a lo siguiente:
 - Bash está instalado en casi todos los sistemas Unix/Linux, por lo que su ejecución no será un problema
 - Hay muchos ejemplos utilizando Bash
- Por lo general los scripts en bash son cortos y realizan una tarea en concreto. Si encuentras que estás haciendo algo muy largo o complejo quizás una mejor opción pueda ser Python o Ruby.

Conceptos básicos de scripting

- Para crear scripts fácilmente es necesario haber interactuado con una Shell, saber algunos comandos básicos, redirecciones y variables.
- Adicionalmente debemos de tener conocimiento de los siguientes conceptos:
 - Tipos de datos avanzados
 - Estructuras de control (condicionales y ciclos)
 - Funciones
 - I/O avanzado ()

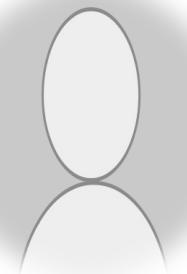


Tipos de datos avanzados

- Adicionalmente a las variables normales de un único valor en un script, es posible almacenar strings



```
# Define an array
os=('Linux' 'macOS' 'windows')
# Print the first element
echo "${os[0]}"
# Get the length of the array
number="#${#os[@]}"
# Print length
echo $number
```

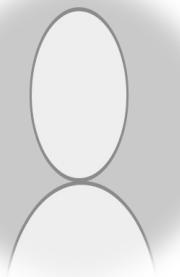


Estructuras de control

```
# Print each file name of a directory
for file in /tmp/* ; do
    echo "$file"
done

# For Range
for i in {1..10}; do
    echo "$i"
done

# Forever while; press Ctrl+C
while true; do
    echo "hello"
    sleep 3
done
```

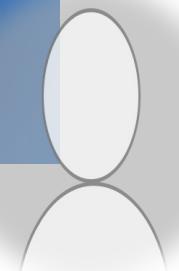


Funciones



```
# Function definition; parameters passed via $n
decirhola() {
    echo "Hello $1, how are you?"
}

# Function invocation
decirhola "Lucas"
```



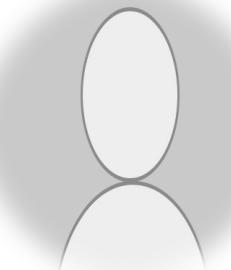
Leyendo entradas de usuario

- Es posible leer la entrada de un usuario desde el stdin con el comando read
- Generalmente los scripts son creados para ejecutados de forma automática sin intervención humana



```
# Read a value from the user input
read name

# Print the value
printf "Hello %s" "$name"
```



Ejecutando Scripts

- Para ejecutar scripts la extensión no es algo que importe, pero por lo general se usa la extensión .sh
- Consideraciones:
 - El script debe de tener permisos de ejecución. Como buena práctica suele utilizarse 750
 - En la primera línea del script se declara el intérprete, que es conocido como shebang

Bash shebang

Es la primera línea de un script que indica que interprete utilizar para ejecutar el script. La directiva toma la siguiente forma:

```
#!interpreter [arguments]
```

- La directiva se coloca en la primera línea del script
- La directiva inicia con el shebang #!
- Un espacio después del shebang es opcional
- El intérprete es el full path de un binario
- Los argumentos del interprete son opcionales

Ejemplos de Shebang

`#!/bin/bash`

- Usa bash para ejecutar el archivo

`#!/usr/bin/env perl`

- Usa el comando env para encontrar el path hacia el ejecutable

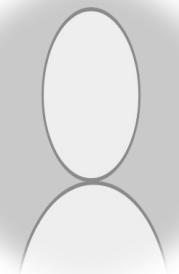
`#!/usr/bin/python`

- Usa Python para ejecutar el archivo

Ejemplo

```
#!/usr/bin/env bash
# If an error is found the script stops
set -o errexit
# Undefined variables are treated as error
set -o nounset
# If a part of the pipe fails, then the whole pipe fails.
set -o pipefail

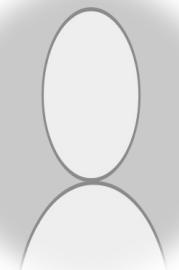
# Parameter with default value
firstargument="${1:-somedefaultvalue}"
echo "$firstargument"
```



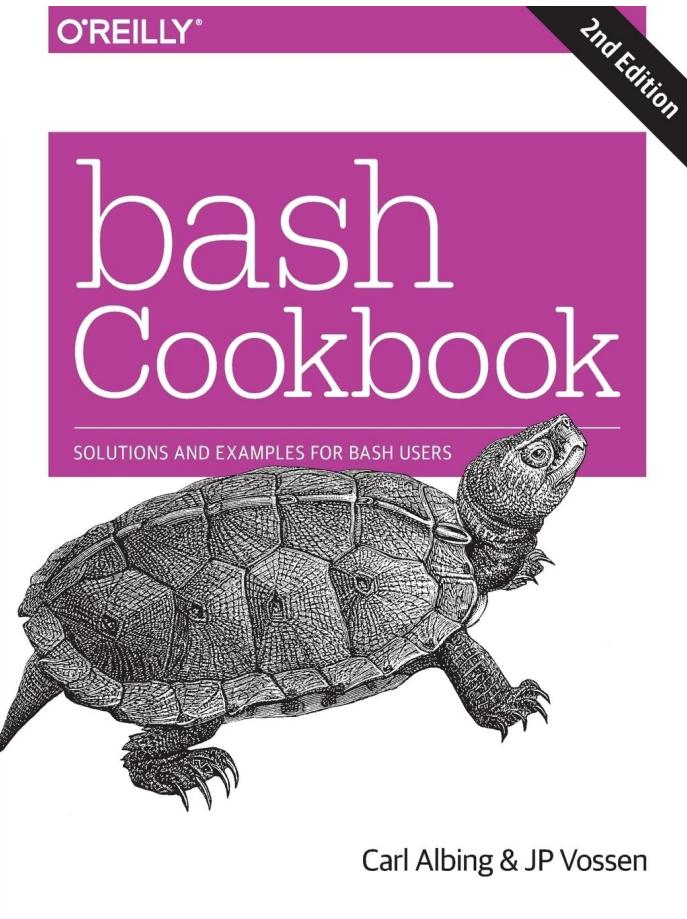


Buenas practicas

- Evitar información sensible
- Sanitización de inputs. Usar defaults
- Chequear dependencias
- Fallar rápido
- Manejo de errores
- Documentar
- Versionamiento
- Pruebas



Mastering Bash



[Bash scripting cheatsheet
\(devhints.io\)](#)

[ShellCheck
, a static analysis tool for shell scripts \(github.com\)](#)

[Bash Reference Manual \(gnu.org\)](#)

Recursos para practicar Bash

- [Bash exercises on Exercism](#)
- [bash tutorial](#)
: Exercises for learning the bash command line. ([github.com](#))
- [https://www.freecodecamp.org/news/shell-scripting-crash-course-h
ow-to-write-bash-scripts-in-linux/](https://www.freecodecamp.org/news/shell-scripting-crash-course-how-to-write-bash-scripts-in-linux/)

Cursos

- <https://www.udemy.com/course/linux-shell-scripting-free/>
- <https://www.udemy.com/course/bash-linux-command-from-scratch/>
- <https://www.coursera.org/projects/introduction-to-bash-shell-scripting>

Actividad 2

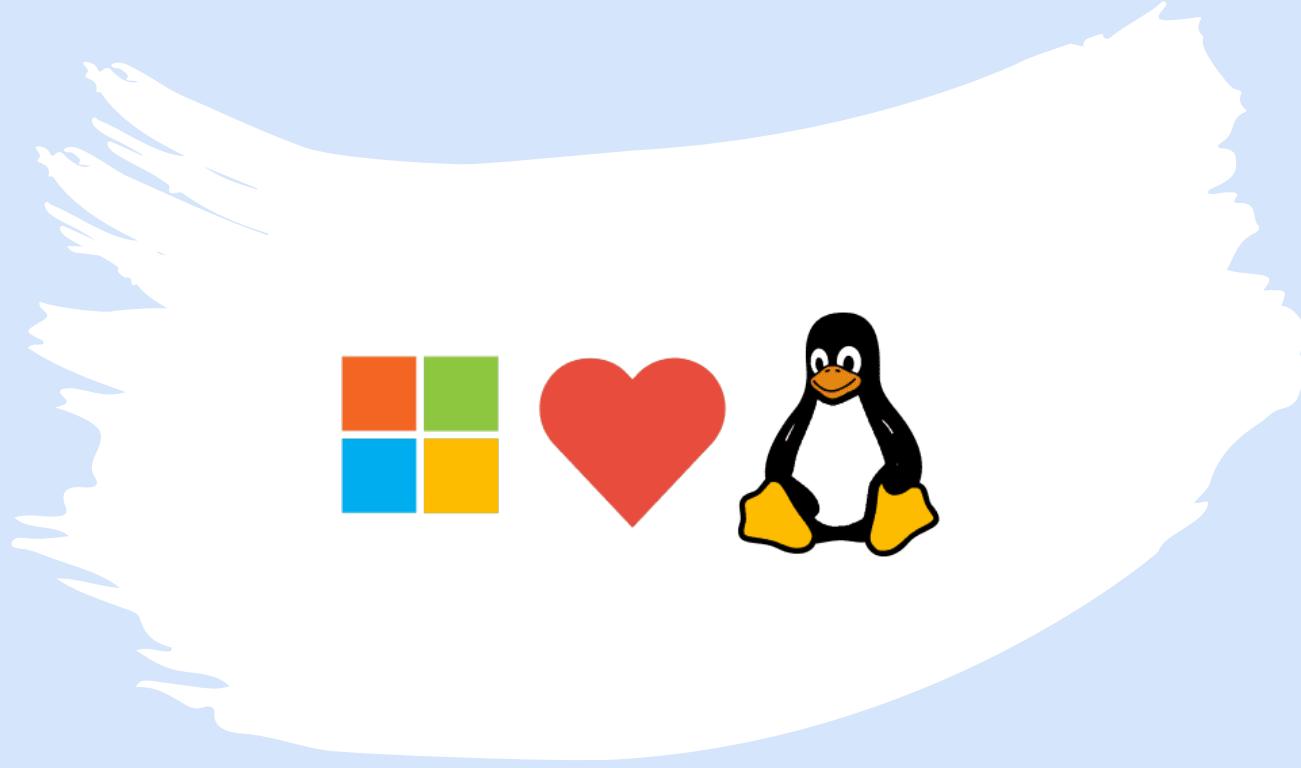
Crear un script de Bash que realice lo siguiente:

- Leer la variable GITHUB_USER
- Consultar la URL <https://api.github.com/users/> concatenando el valor de la variable GITHUB_USER al final
- Imprimir el mensaje
 - "Hola <github_user>. User ID: <id>. Cuenta fue creada el: <created_at>."
- Crear un log file en /tmp/<fecha>/saludos.log con la salida del mensaje anterior. Donde <fecha> corresponde a la fecha del día de ejecución del script
- Crear un cronjob para que el script se ejecute cada 5 minutos

Fecha de entrega: 28/07/24

Entrega: Link al folder de GitHub

Nombre del folder: **actividad2**



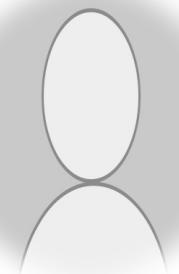
Muchas Gracias

Universidad de San Carlos de Guatemala
Facultad de Ingeniería
Escuela de Ciencias y Sistemas
Sistemas Operativos



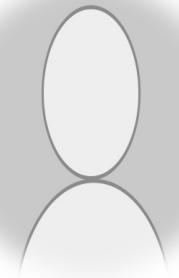
Access Control

Usuarios, grupos y permisos en Linux



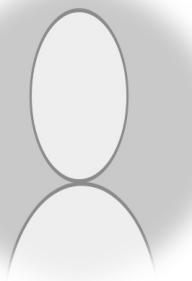


-
- En un sistema operativo multi usuario, ¿cuáles son los permisos que tengo sobre los recursos y que acciones puedo realizar?
 - ¿Cuál es la relación entre un usuario, un archivo y los procesos ?
 - ¿Como puedo gestionar esos permisos ?
 - El control de acceso a los recursos es una parte vital de un sistema multiusuario



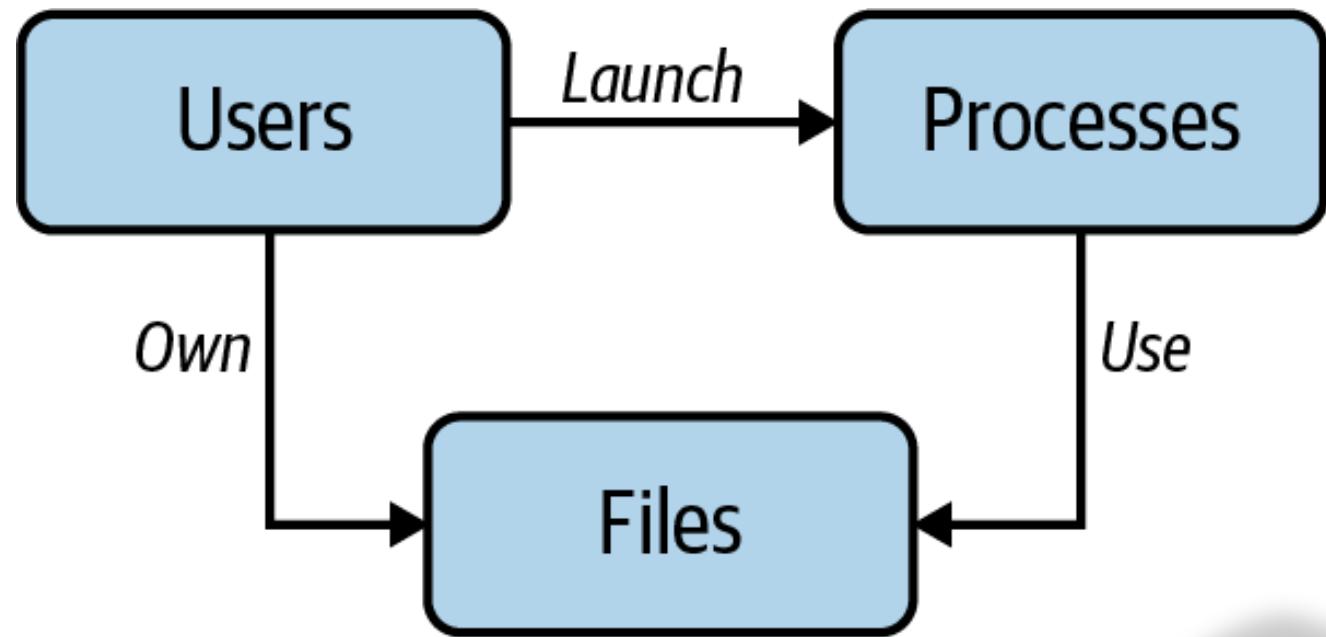
Recursos y propiedad

- Linux es un sistema multiusuario como Unix
- Una cuenta de usuario es identificada con un UID
- Los recursos (files) son propiedad de un usuario
- Por lo general, un usuario humano utiliza una cuenta de usuario para loguearse y ejecutar procesos



Relación entre usuarios, archivos y procesos en Linux

- Un usuario tiene archivos e inicializa procesos
- Los archivos tienen propietarios; generalmente el usuario que lo crea
- Los procesos usan archivos para persistencia y comunicación. Los usuarios usan los archivos de forma indirecta por medio de un proceso



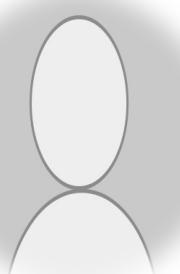
Tipos de Control de Acceso

Control de acceso discrecional (DAC)

- La idea es bloquear el acceso a los recursos basados en la identidad de un usuario
- Es discrecional porque un usuario con permisos puede otorgalo a otro usuario
- Este es el enfoque más común

Control de acceso mandatorio (MAC)

- Esta basando en un sistema jerárquico que representa niveles de seguridad
- La validación de acceso se hace a nivel de Kernel por lo que tiene prioridad sobre el acceso discrecional
- Es mandatorio porque es el administrador del SO que lo gestiona
- Se implementa con SELinux o AppArmor



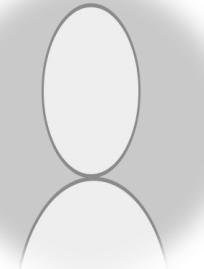


A collection of wooden human-shaped cutouts, known as "paper dolls," are scattered across the right side of the slide. They are painted in various colors including red, blue, green, and brown. The background is a solid dark grey or black, making the colorful figures stand out.

Usuarios

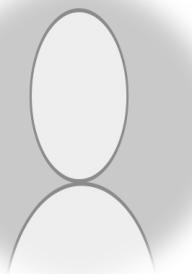
Usuarios

- En Linux por lo general tenemos dos tipos de usuario:
 - System Users
 - Típicamente programas que utilizan este usuario para correr procesos en background (mysql, apache, etc.)
 - Regular Users
 - Un usuario humano que interactúa con Linux usando una Shell.
- Los usuarios son identificados con UID y pertenecen a uno o más grupos identificados como GID
- Existe un usuario especial con el UID 0, que generalmente es llamado root. Este usuario tiene permisos para ejecutar cualquier tipo de acción.



Conversión para los UIDs en systemd

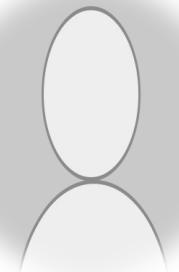
- UID 0
 - Usuario root
- UID 1 a 999
 - Reservado para system users
- UID 65534
 - Es el usuario nobody. Generalmente usado en mapeo de Network File Systems
- UID 1000 a 65533 y 65536 a 4294967294
 - Para regular users
- Es posible obtener el UID de tu usuario ejecutando el comando id



Manejando usuarios localmente

- Es la opción tradicional y la única disponible por defecto
- La información de los usuarios es almacenada en la maquina
- Linux usa archivos para almacenar la configuracion de los usuarios

Purpose	File
User database	<i>/etc/passwd</i>
Group database	<i>/etc/group</i>
User passwords	<i>/etc/shadow</i>
Group passwords	<i>/etc/gshadow</i>



El archivo /etc/passwd

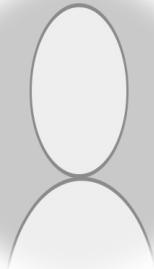
- #1 el usuario root
- #2 un system user
- #3 un regular user

```
cat /etc/passwd
# File Content
root:x:0:0:root:/root:/bin/bash # 1
daemon:x:1:1:daemon:/usr/sbin:/usr/sbin/nologin # 2
bin:x:2:2:bin:/bin:/usr/sbin/nologin
sys:x:3:3:sys:/dev:/usr/sbin/nologin
nobody:x:65534:65534:nobody:/nonexistent:/usr/sbin/nologin
syslog:x:104:110::/home/syslog:/usr/sbin/nologin
mh9:x:1000:1001::/home/mh9:/usr/bin/fish # 3
```

Detalle un usuario

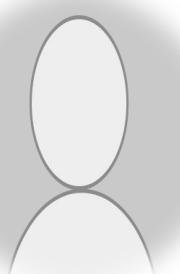
```
root:x:0:0:root:/root:/bin/bash
^   ^ ^ ^ ^ ^   ^   ^
|   | | | | |   |   |
|   | | | | |   |   | ①
|   | | | | |   |   | ②
|   | | | | |   |   | ③
|   | | | | |   |   | ④
|   | | | | |   |   | ⑤
|   | | | | |   |   | ⑥
|   | | | | |   |   | ⑦
```

1. La Shell a utilizar
2. El home directory
3. Información adicional. Campo **GECOS**.
4. El grupo principal GID. Ver */etc/group*
5. El UID
6. El password. Una x significa que este encriptado (*/etc/shadow*)
7. El nombre del usuario



Gestionando usuarios

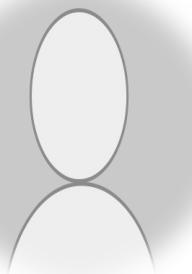
- El comando adduser/useradd nos ayuda a gestionar la creación de usuarios
- Es posible utilizar el flag -r para crear un system user
- El comando passwd se utiliza para configurar el password de un usuario
- El comando usermod se utiliza para configurar un usuario
- Por defecto cuando se crea un usuario se crea un grupo con le mismo nombre
- El comando groupadd nos ayuda a gestionar la creación de grupos
- El comando gpasswd se utiliza para configurar el password de un grup



Gestión de usuarios centralizada

Por lo general en sistemas empresariales se suelen administrar múltiples computadoras por lo que es común tener una gestión de usuarios centralizada. Algunos enfoques comunes son los siguientes:

- Basados en directorios
 - Por lo general se utiliza el protocolo LDAP. E.g. [Keycloak](#) o Microsoft Active Directory
- Vía Network
 - Los usuarios pueden autenticarse de esta forma usando [Kerberos](#)
- Usando sistemas de gestión de configuración
 - Estos son sistemas como Puppet, Ansible, Chef que pueden ser usados para crear usuarios en muchas computadoras

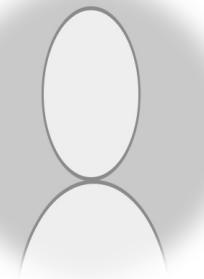


Permisos



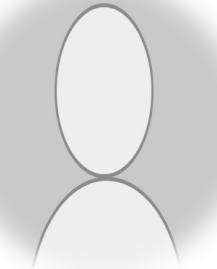
Permisos de archivo

- El acceso a los archivos es el core de los permisos de Linux, puesto que “todo es un archivo en Linux”
- Existen tres tipos de **scopes** de permisos, desde lo más pequeño a lo más grande:
 - Usuario
 - El propietario del archivo
 - Grupo
 - Tiene un o más miembros
 - Otro
 - La categoría para todo lo demás



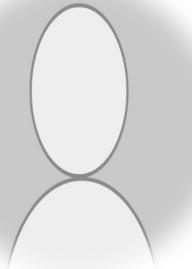
Tipos de acceso

- Read (r)
 - Para un archivo, permite ver el contenido. Para un directorio, permite ver el nombre de los archivos en el directorio
- Write (w)
 - Para un archivo, permite la modificación o eliminación del archivo. Para un directorio, permite crear, renombrar y borrar archivos en el directorio.
- Execute (x)
 - Para un archivo, permite ejecutar el archivo si el usuario tiene permiso de lectura. Para un directorio, permite acceder a la información de los archivos y listar su contenido.



Otros File Access Bits

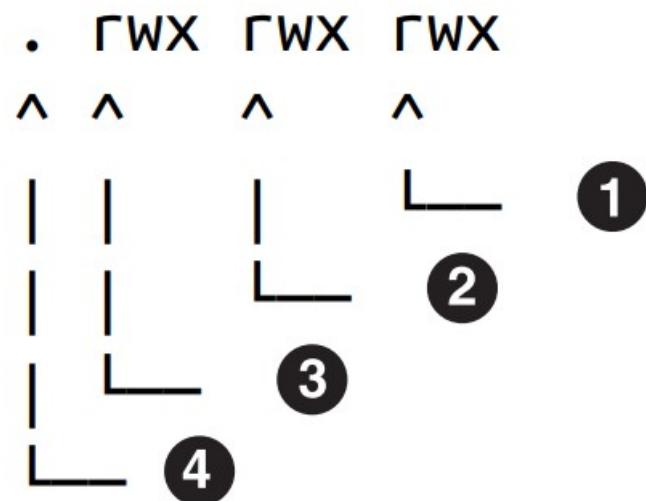
- Setuid/setgid (s)
 - Aplicado a los archivos ejecutables. Un usuario ejecutándolo hereda los permisos del propietario o grupo del archivo
- Sticky bit (t)
 - Es solo relevante para directorios. Si se establece, evita que los usuarios no root puedan eliminar archivos en el directorio, a menos que dicho usuario sea propietario del directorio/archivo.



Viendo permisos de un archivo

1. El nombre del archivo
 2. La fecha de modificación
 3. El tamaño en bytes
 4. El grupo al que pertenece el archivo
 5. El propietario
 6. El número de hardlinks
 7. El modo del archivo
(Permisos)

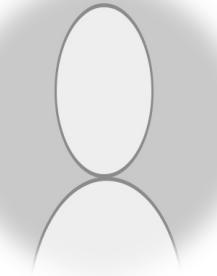
El file mode



1. Permisos para otros
2. Permisos para el grupo
3. Permisos del propietario
4. El tipo de archivo

Tipos de archivos en el mode

Symbol	Semantics
-	A regular file (such as when you do <code>touch abc</code>)
b	Block special file
c	Character special file
c	High-performance (contiguous data) file
d	A directory
l	A symbolic link
p	A named pipe (create with <code>mkfifo</code>)
s	A socket
?	Some other (unknown) file type

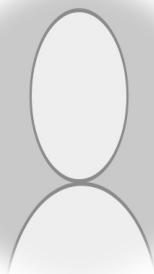


Representación decimal de los permisos

Pattern	Effective permission	Decimal representation
---	None	0
--x	Execute	1
-w-	Write	2
-wx	Write and execute	3
r--	Read	4
r-x	Read and execute	5
rw-	Read and write	6
rwx	Read, write, execute	7

Ejemplos:

- 755
 - Todos los permisos para el propietario. Lectura y ejecución para los demás
- 700
 - Todos los permisos para el propietario. Ninguno para los demás
- 644
 - Lectura y escritura para el propietario. Solo lectura para los demás
- 400
 - Solo lectura por el propietario



Gestionando permisos

```
# List the permissions
$ ls -al /tmp/masktest
-rw-r--r-- 1 mh9 dev 0 Aug 28 13:07 /tmp/masktest

# Change permissions. Make file executable
$ chmod +x /tmp/masktest

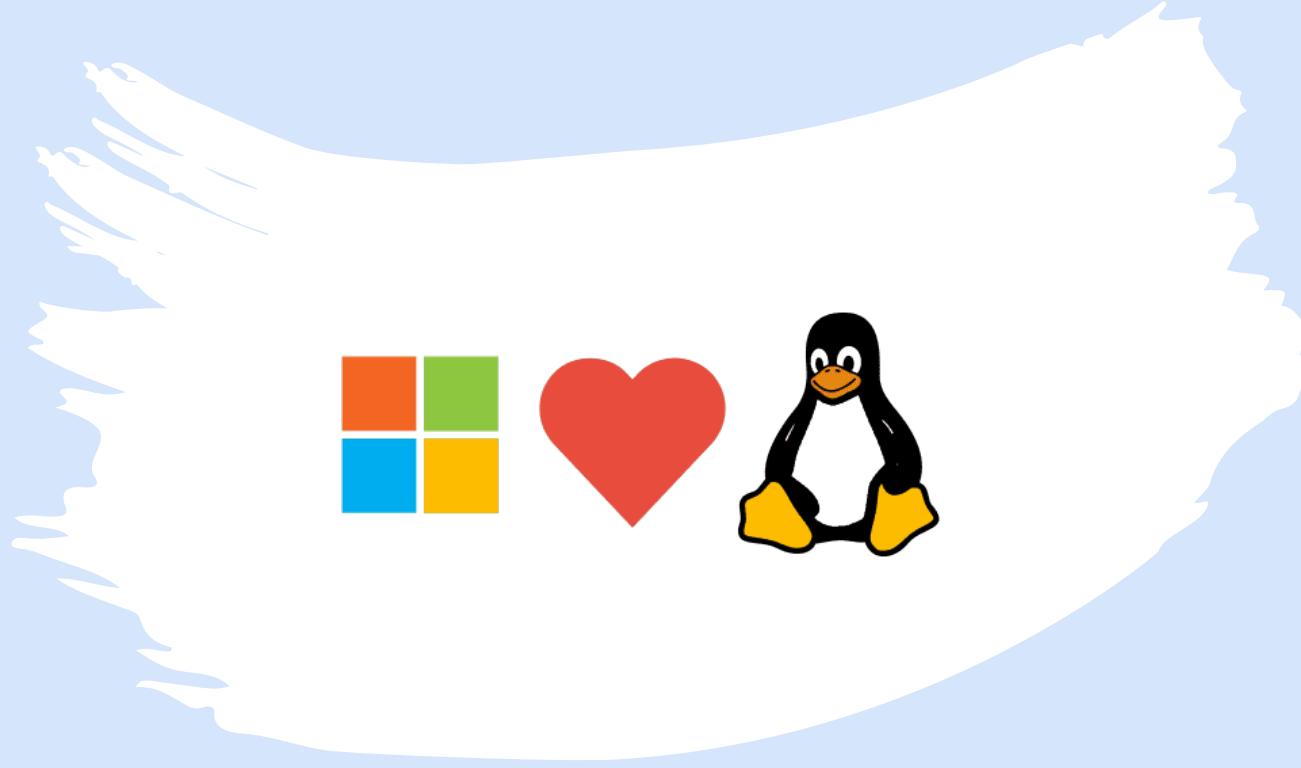
# List permissions
$ ls -al /tmp/masktest
-rwxr-xr-x 1 mh9 dev 0 Aug 28 13:07 /tmp/masktest

# Make file executable only for the owner
$ chmod 744 /tmp/masktest

# List permissions
$ ls -al /tmp/masktest
-rwrxr--r-- 1 mh9 dev 0 Aug 28 13:07 /tmp/masktest
```

Owner	Group	Other
r W - 1 1 0 6	r - - 1 0 0 4	r - - 1 0 0 4
chmod +x		
r W X 1 1 1 7	r - X 1 0 1 5	r - X 1 0 1 5

Note: también es posible utilizar el comando chown para cambiar el propietario y grupo de un archivo



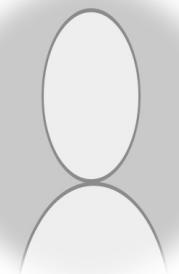
Muchas Gracias

Universidad de San Carlos de Guatemala
Facultad de Ingeniería
Escuela de Ciencias y Sistemas
Sistemas Operativos



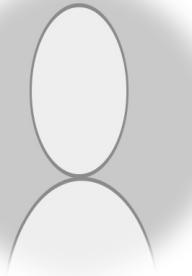
Filesystems

Archivos en Linux



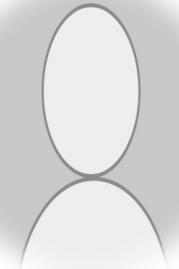
Conceptos básicos

Antes de empezar

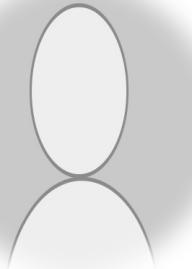


Conceptos básicos

- La mayoría de los filesystem son jerárquicos. Estos proveen al usuario un árbol para el filesystem que empieza en /.
- En los árboles de filesystem hay dos tipos de objetos diferentes: archivos y directorios. Los directorios ayudan a organizar y agrupar archivos.
- Es posible navegar dentro de un filesystem listando el contenido de un folder con ls, cambiando de directorio con cd o revisando el directorio actual con pwd
- Generalmente los filesystem son implementados en el Kernel por temas de rendimiento, pero también hay una opción para implementarlo en el user land usando [FUSE](#).

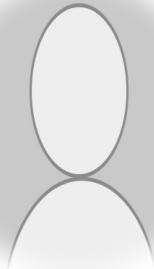


- **Drive o Device.** Un dispositivo de bloques físico como un HDD o un SSD. Estos pueden encontrarse en el folder /dev. E.j. /dev/sda
- **Partición.** Es una separación lógica de un device. E.j. si se decide partir un disco en dos particiones se tendrá /dev/sda1 y /dev/sda2
- **Volumen.** Es similar a una partición, pero más flexible permitiendo unir múltiples devices.
- **Super Block.** Una sección especial del filesystem que contiene metadata como el tipo de filesystem, inodos, bloques, estado, etc.
- **Inodos.** En un filesystem, los inodos guardan metadata de los archivos como ubicación, tamaño, permisos, etc.



Comandos de Filesystem y block devices

Command	Use case
<code>lsblk</code>	List all block devices
<code>fdisk, parted</code>	Manage disk partitions
<code>blkid</code>	Show block device attributes such as UUID
<code>hwinfo</code>	Show hardware information
<code>file -s</code>	Show filesystem and partition information
<code>stat, df -i, ls -i</code>	Show and list inode-related information



Links

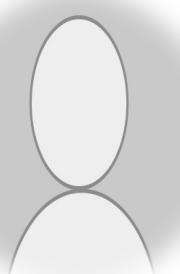
Son tipos especiales de archivos que se utilizan para hacer referencia a otros archivos.

- Hard Links

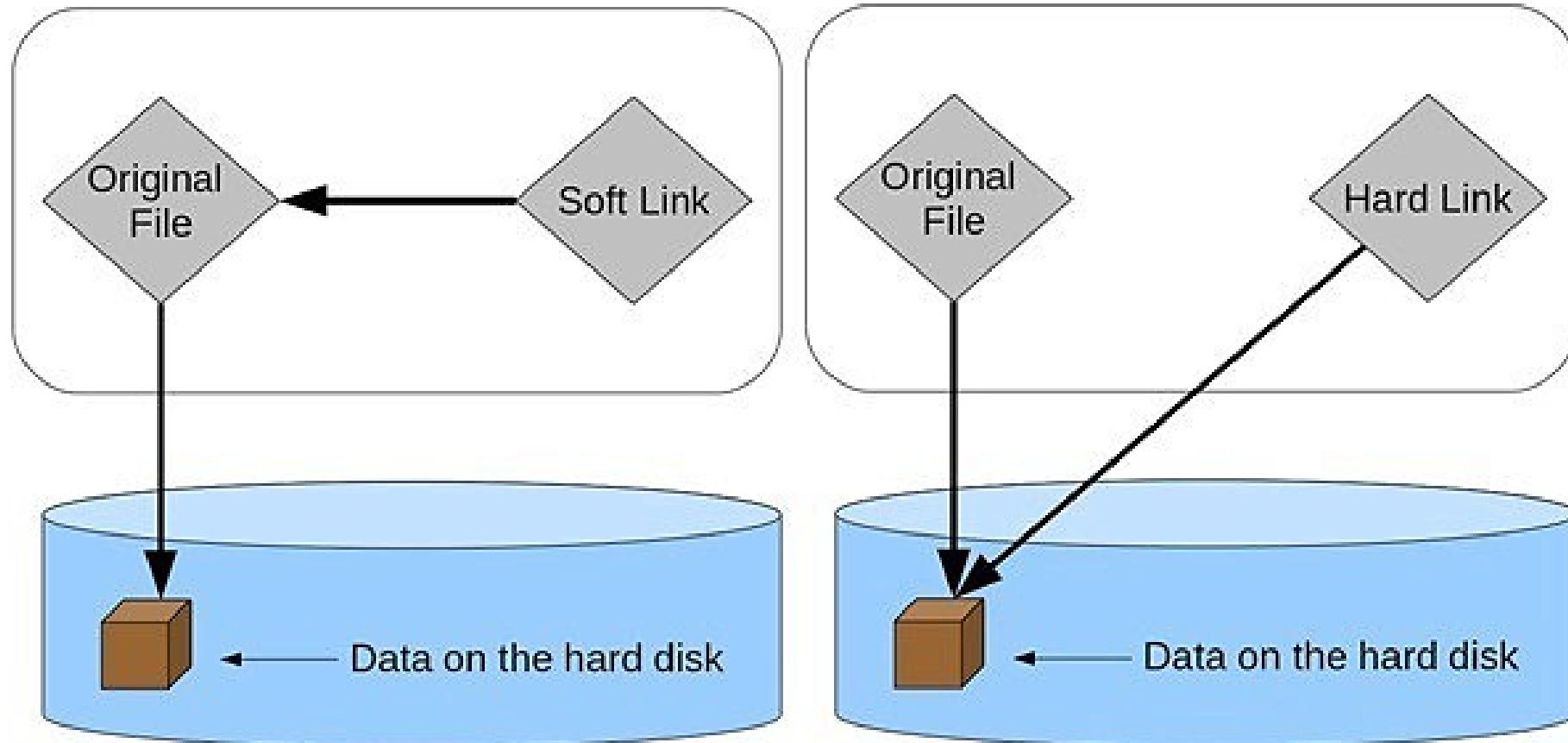
- Hace referencia a un inodo (otro file). Una vez creado puede ser tratado como un archivo independiente
- No puede hacer referencia a un directorio u otro filesystem

- Symbolic Links o symlinks

- Son un archivo especial que en su contenido tiene el path hacia otro archivo
- Funcionan como acceso directo al archivo original
- Si el archivo original es eliminado el link simbólico no funcionara



Hard Link and Soft Link

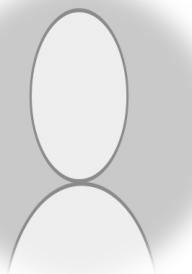


Interactuando con Links

```
$ ln myfile somealias # creates a hard link
$ ln -s myfile somesoftalias # creates a soft link

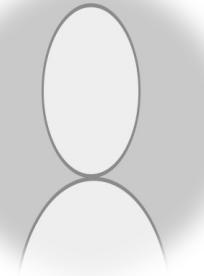
$ ls -al *alias
-rw-rw-r-- 2 mh9 mh9 0 Sep 5 12:15 somealias
lrwxrwxrwx 1 mh9 mh9 6 Sep 5 12:45 somesoftalias → myfile

$ stat somealias
  File: somealias
  Size: 0 Blocks: 0 IO Block: 4096 regular empty file
Device: fd00h/64768d Inode: 6302071 Links: 2
...
$ stat somesoftalias
  File: somesoftalias → myfile
  Size: 6 Blocks: 0 IO Block: 4096 symbolic link
Device: fd00h/64768d Inode: 6303540 Links: 1
```



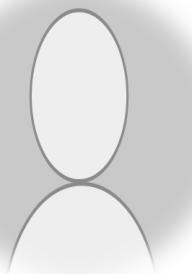
The Virtual File System

“Todo es un archivo en Linux”



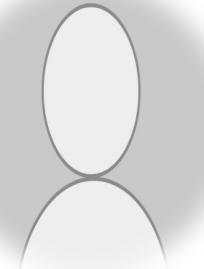
The Virtual File System

- Es una abstracción el Kernel que provee a los clientes acceder a los recursos como si fueran archivos.
- Linux no define la estructura de un archivo; es únicamente un stream de bytes que el cliente de debe de interpretar
- La idea básica es tener un intermediario entre los recursos y la llamada de sistema de archivos

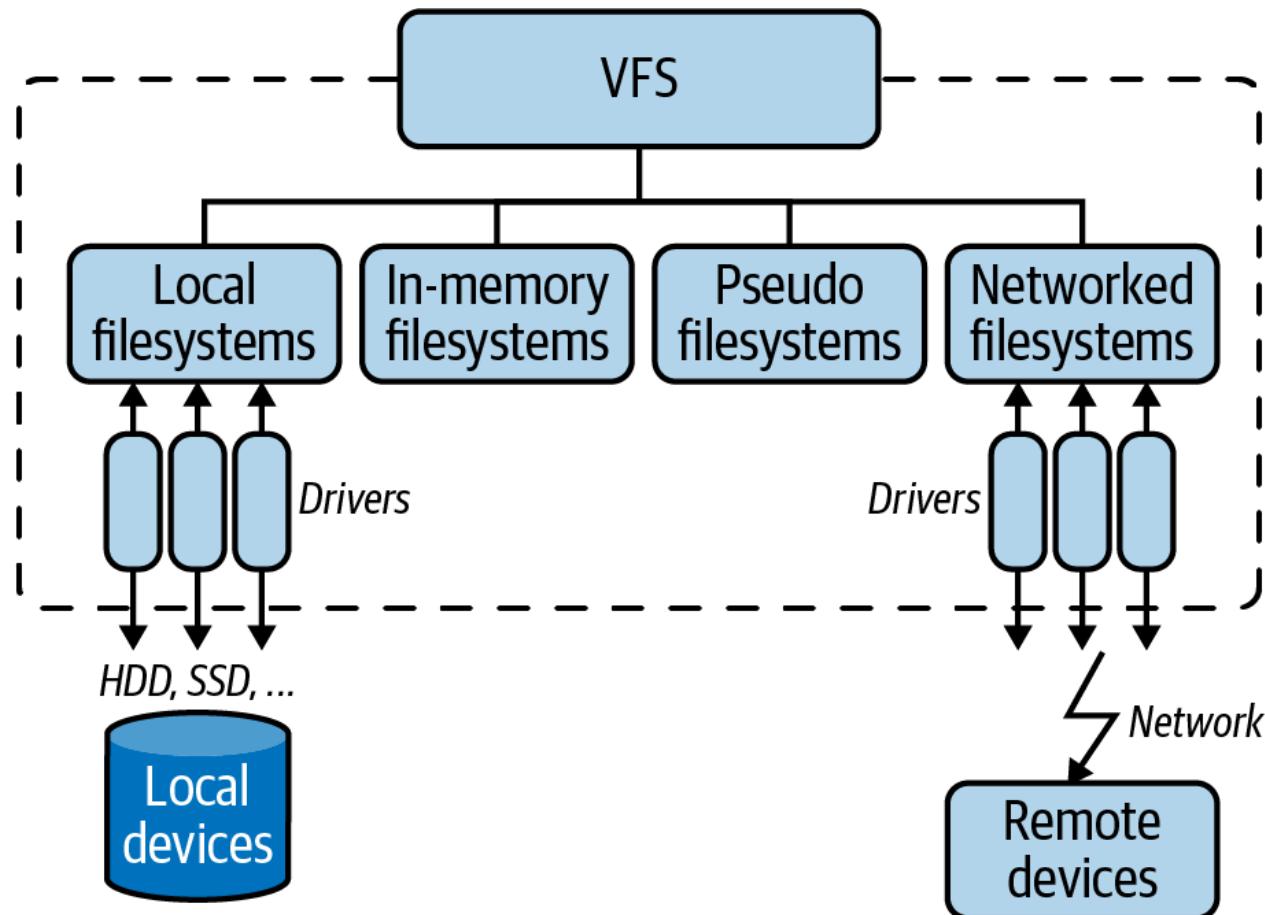


Abstracciones de VFS

- **Local filesystems**, como ext3, XFS, FAT, y NTFS
 - Estos filesystems usan drivers para acceder a dispositivos de bloques como HDDs o SSDs.
- **In-memory filesystems**, como tmpfs,
 - Estos viven en la memoria principal (RAM)
- **Pseudo filesystems**, como procfs,
 - Estos filesystems también viven en memoria, pero son usados como interfaces de Kernel y devices.
- **Networked filesystems**, como NFS o Samba
 - Estos también usan un driver; sin embargo, estos devices no se encuentran localmente

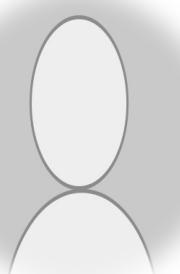


Abstracciones de VFS



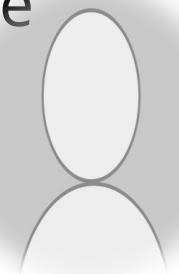
Logical Volume Manager (LVM)

- LVM es una capa intermedia entre entidades físicas (como unidades o particiones) y el sistema de archivos para permitir una gestión más flexible y eficiente
- Permite con un riesgo bajo y sin downtime expandir volúmenes

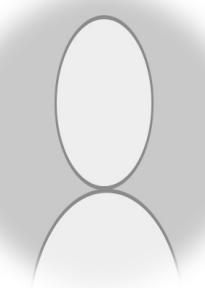
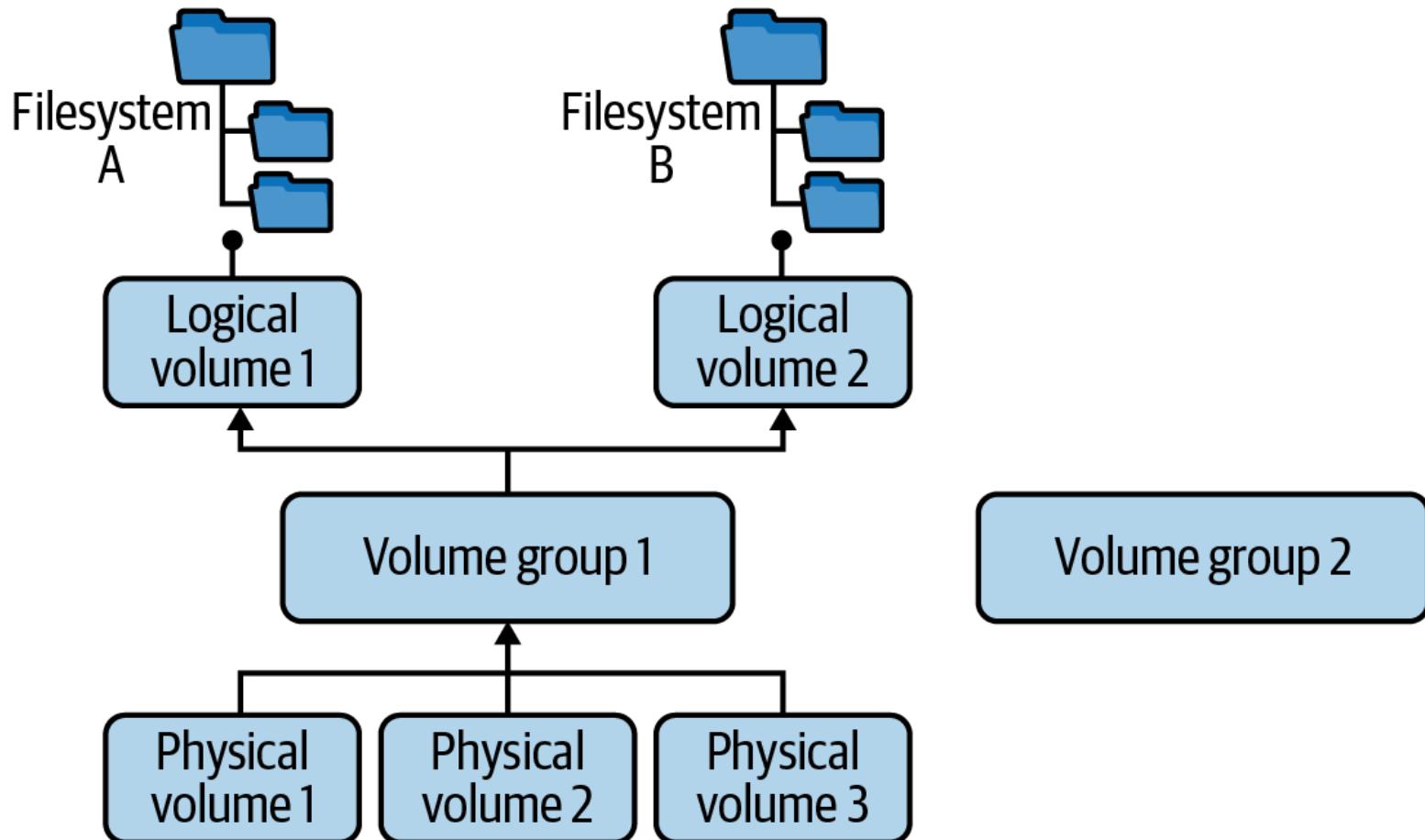


Componentes de LVM

- Physical volumes (PV)
 - Pueden ser una partición de disco, un disco entero y otros dispositivos.
- Volume groups (VG)
 - Son intermediarios entre un conjunto de PVs y LVs. Piense en un VG como un conjunto de PVs que proporcionan recursos colectivamente.
- Logical volumes (LV)
 - Son dispositivos de bloque creados a partir de VGs. Son conceptualmente comparables a las particiones. Debe crear un filesystem en un LV antes de poder usarlo. Puede cambiar fácilmente el tamaño de los LV mientras están en uso.

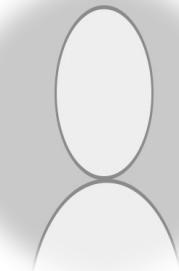


Linux LVM overview



Comandos para LVM

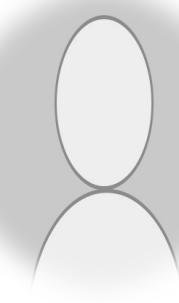
- Tools para PV
 - lvmdiskscan
 - pvdisplay
 - pvcreate
 - pvscan
- Tools para VG
 - vgs
 - vgdisplay
 - vgcreate
 - vgextend
- Tools para LV
 - lvs
 - lvscan
 - Lvcreate
 - lvextend



Operaciones de Filesystem

Las operaciones comunes con un filesystem son las siguientes:

- Crear un filesystem.
 - En sistemas operativos no Linux este proceso es conocido como formateo
 - Puede realizarse utilizando una partición o un logical volume
 - El comando a utilizar es mkfs -t <filesystem type> <Partiotion or LV>.
 - Ejemplo mkfs -t ext4 /dev/some_vg/some_lv
- Montar un filesystem
 - El montaje se refiere a la disponibilidad de este el árbol del filesystem
 - El comando para montar o listar los montajes es mount

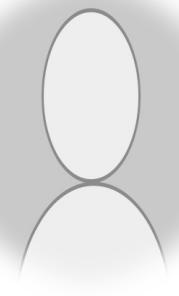


Layout común de los filesystems

- Una vez creado un filesystem la pregunta común es estructura u organización debemos de utilizar para almacenar los archivos (programas, archivos de configuración, librerías)
- A esta organización de directorios se le conoce como filesystem layout. Formalmente [Filesystem Hierarchy Standard \(FHS\)](#).
- Esta define directorios, estructura y su contenido.
- A pesar de que FHS es mantenido por Linux Foundation, son las distribuciones las que deciden que estructura utilizar.

Top Level de directorios comunes

Directory	Semantics
bin, sbin	System programs and commands (usually links to /usr/bin and /usr/sbin)
boot	Kernel images and related components
dev	Devices (terminals, drives, etc.)
etc	System configuration files
home	User home directories
lib	Shared system libraries
mnt	media Mount points for removable media (e.g., USB sticks)
opt	Distro specific; can host package manager files
proc	sys Kernel interfaces
tmp	For temporary files
usr	User programs (usually read-only)
var	User programs (logs, backups, network caches, etc.)



Pseudo Filesystems

Pseudo Filesystems

- Los pseudo Filesystems están exponiendo ciertas interfaces del Kernel, pero en forma de Filesystem.
- Por lo tanto, los pseudo Filesystems actúan como Filesystems por lo que es posible interactuar con ellos con ls, cd, cat.
- Algunas de las interfaces que exponen:
 - Información de procesos
 - Interacciones con dispositivos como teclados
 - Utilidades como dispositivos especiales

procfs

- Linux hereda el /proc Filesystem de Unix. La función original era publicar información de los procesos para que fueran accedidos con ps y free.
- Es un Filesystem read only que tiene pocas reglas sobre su estructura, pero por lo general se puede encontrar la siguiente información:
 - Información de cada proceso en /proc/ID, donde el ID es el identificador del proceso
 - Información varia como mounts, información relacionada a redes, información de memoria, versión del sistema y update.

Informació n por proceso en procfs

Entry	Type	Information
attr	Directory	Security attributes
cgroup	File	Control groups
cmdline	File	Command line
cwd	Link	Current working directory
environ	File	Environment variables
exe	Link	Executable of the process
fd	Directory	File descriptors
io	File	Storage I/O (bytes/char read and written)
limits	File	Resource limits
mem	File	Memory used
mounts	File	Mounts used
net	Directory	Network stats
stat	File	Process status
syscall	File	Syscall usage
task	Directory	Per-task (thread) information
timers	File	Timers information

sysfs

- El Filesystem sysfs es específico a Linux y es una forma estructurada del Kernel para exponer información, comúnmente de dispositivos, usando un layout estandarizado
- Cierta información que está en sysfs también está disponible en procfs, pero también hay información que solo está disponible en procfs (como información de la memoria)
- El layout principal o raíz se encuentra bien documentado y estandarizado pero las carpetas internas puede que no lo estén

Layout de sysfs

block/

- Directorio con soft links a dispositivos de bloques descubridos.

bus/

- Contiene un subdirectorio por cada bus físico en el Kernel.

class/

- Contiene las clases de los dispositivos.

dev/

- Contiene dos subdirectorios: block/ para block devices y char/ para dispositivos de caracteres.

devices/

- En este directorio, el núcleo proporciona una representación del árbol de dispositivos.

firmware/

- A través de estos directorios, puede administrar atributos específicos del firmware.

fs/

- Este directorio contiene subdirectorios para algunos sistemas de archivos.

module/

- En estos directorios encontrará subdirectorios para cada módulo cargado en el kernel.

devfs

- El Filesystem /dev (devfs) contiene información sobre archivos especiales sobre devices
- Contiene información de dispositivos físicos, pero también contiene otras cosas como generadores de números aleatorios.
- Los dispositivos disponibles y gestionados a través de devfs son:

Devices disponible s y gestionad os por devfs

Block devices

- Manejar datos en bloques, por ejemplo, dispositivos de

Character devices

- Manejar las cosas carácter por carácter, como una terminal, un

Dispositivos especiales

- Generar datos o permitir manipularlos, como los famosos /dev/null o /dev/random

Archivos regulares

Filesystem comunes

Filesystem	Linux support since	File size	Volume size	Number of files	Filename length
ext2	1993	2 TB	32 TB	10^{18}	255 characters
ext3	2001	2 TB	32 TB	variable	255 characters
ext4	2008	16 TB	1 EB	4 billion	255 characters
btrfs	2009	16 EB	16 EB	2^{18}	255 characters
XFS	2001	8 EB	8 EB	2^{64}	255 characters
ZFS	2006	16 EB	2^{128} Bytes	10 ¹⁴ files per directory	255 characters
NTFS	1997	16 TB	256 TB	2^{32}	255 characters
vfat	1995	2 GB	N/A	2^{16} per directory	255 characters

Filesystem populares

ext4

- Utilizado por defecto en muchas distribuciones hoy en día. Es una evolución compatible con versiones anteriores de ext3. Al igual que ext3, ofrece journaling, es decir, los cambios se registran en un registro para que, en el peor de los casos, la recuperación sea rápida. Una buena opción de propósito general.

XFS

- Un sistema de archivos de diario que fue diseñado originalmente por Silicon Graphics (SGI) para sus estaciones de trabajo a principios de 1990. Al ofrecer soporte para archivos grandes y I/O de alta velocidad, ahora se usa, por ejemplo, en la familia de distribuciones de Red Hat.

ZFS

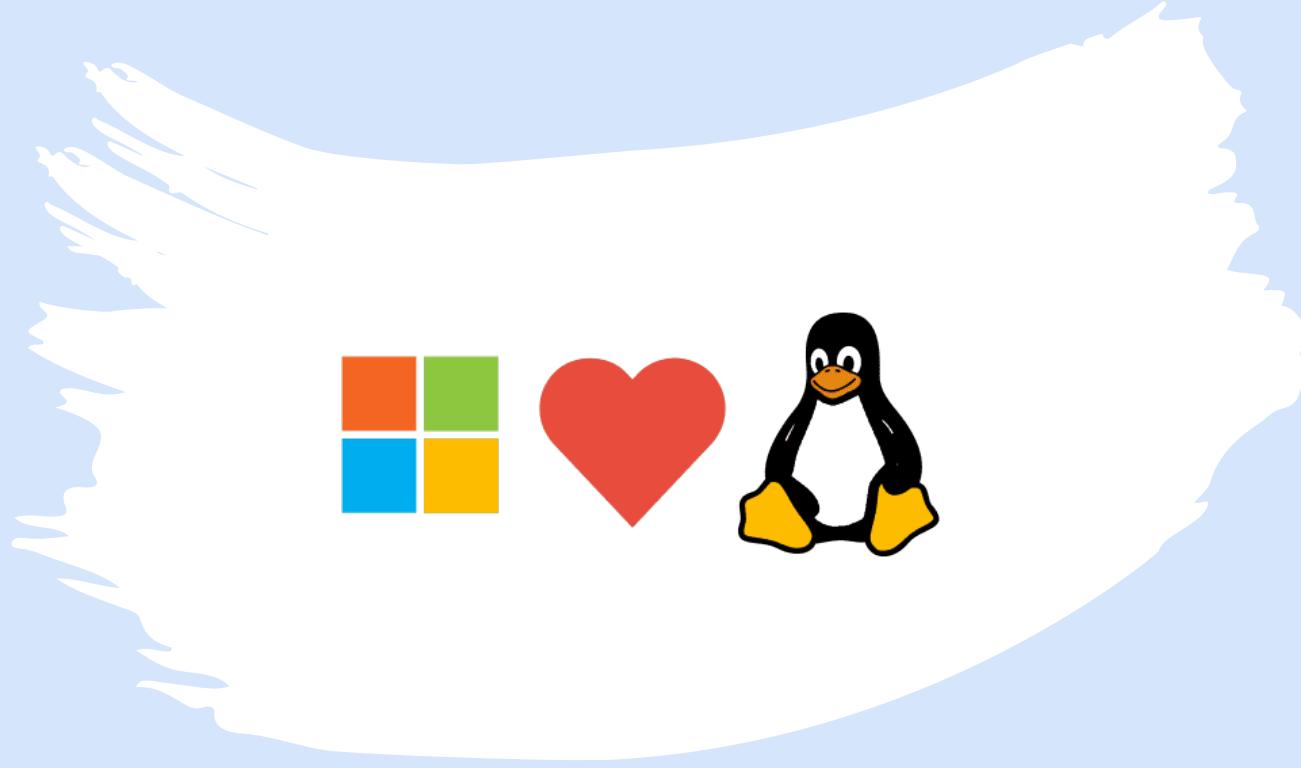
- Desarrollado originalmente por Sun Microsystems en 2001, ZFS combina la funcionalidad del sistema de archivos y el administrador de volúmenes. Si bien ahora existe el proyecto OpenZFS, que ofrece un camino a seguir en un contexto de código abierto, existen algunas preocupaciones sobre la integración de ZFS con Linux.

FAT

- Esta es realmente una familia de sistemas de archivos FAT para Linux, y vfat se usa con mayor frecuencia. El principal caso de uso es la interoperabilidad con los sistemas Windows, así como con los medios extraíbles que utilizan FAT.

Filesystems en memoria

- **debugfs**
 - Un sistema de archivos de propósito especial utilizado para la depuración; generalmente montado con
 - `mount -t debugfs none /sys/kernel/debug`.
- **loops**
 - Permite mapear un sistema de archivos a bloques en lugar de dispositivos.
- **pipefs**
 - Un (pseudo) sistema de archivos especial montado en pipe: que habilita pipes.
- **sockfs**
 - Otro (pseudo) sistema de archivos especial que hace que los sockets de red parezcan archivos, ubicados entre las llamadas al sistema y los sockets.
- **swapfs**
 - Se utiliza para realizar intercambios (no montable).
- **tmpfs**
 - Un sistema de archivos de propósito general que mantiene los datos de los archivos en las memorias caché del Kernel. Es rápido, pero no persistente (apagado significa que se pierden datos).

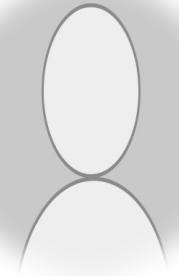


Muchas Gracias

Universidad de San Carlos de Guatemala
Facultad de Ingeniería
Escuela de Ciencias y Sistemas
Sistemas Operativos

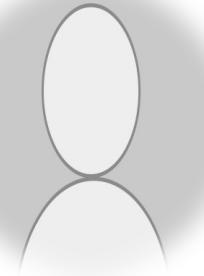


Aplicaciones en Linux



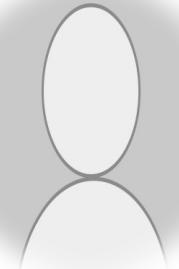
Conceptos básicos

Recap sobre conceptos de procesos y aplicaciones



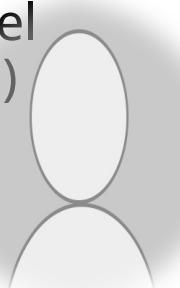
Conceptos Básicos

- Program
 - Este suele ser un archivo binario o un script de shell que Linux puede cargar en la memoria y ejecutar. Otra forma de referirse a esta entidad es ejecutable.
- Process
 - Una entidad en ejecución basada en un programa, cargada en la memoria principal y que usa la CPU o la I/O, cuando no está en reposo.
- Daemon
 - A veces llamado servicio, es un proceso en segundo plano que proporciona una determinada función a otros procesos.
- Application
 - Un programa incluyendo sus dependencias. Por lo general, un programa sustancial, que incluye una interfaz de usuario.

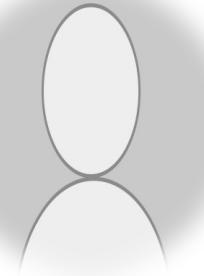


Conceptos Básicos

- Package
 - Un archivo que contiene programas y configuraciones; Se utiliza para distribuir aplicaciones de software.
- Package Manager
 - Un programa que toma un paquete como entrada y, según su contenido y las instrucciones del usuario, lo instala, lo actualiza o lo elimina de un entorno Linux.
- Supply Chain
 - Una colección de productores y distribuidores de software que le permiten encontrar y usar aplicaciones basadas en paquetes.
- Booting
 - La secuencia de inicio en Linux que implica pasos de inicialización del hardware y del SO, incluida la carga del Kernel y el lanzamiento de programas de servicio (daemon) con el objetivo de llevar a Linux a un estado en el que se pueda usar.

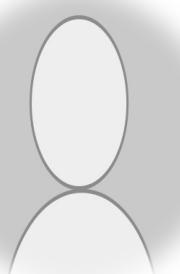


The Linux Startup Process

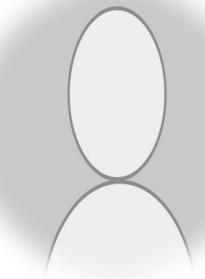
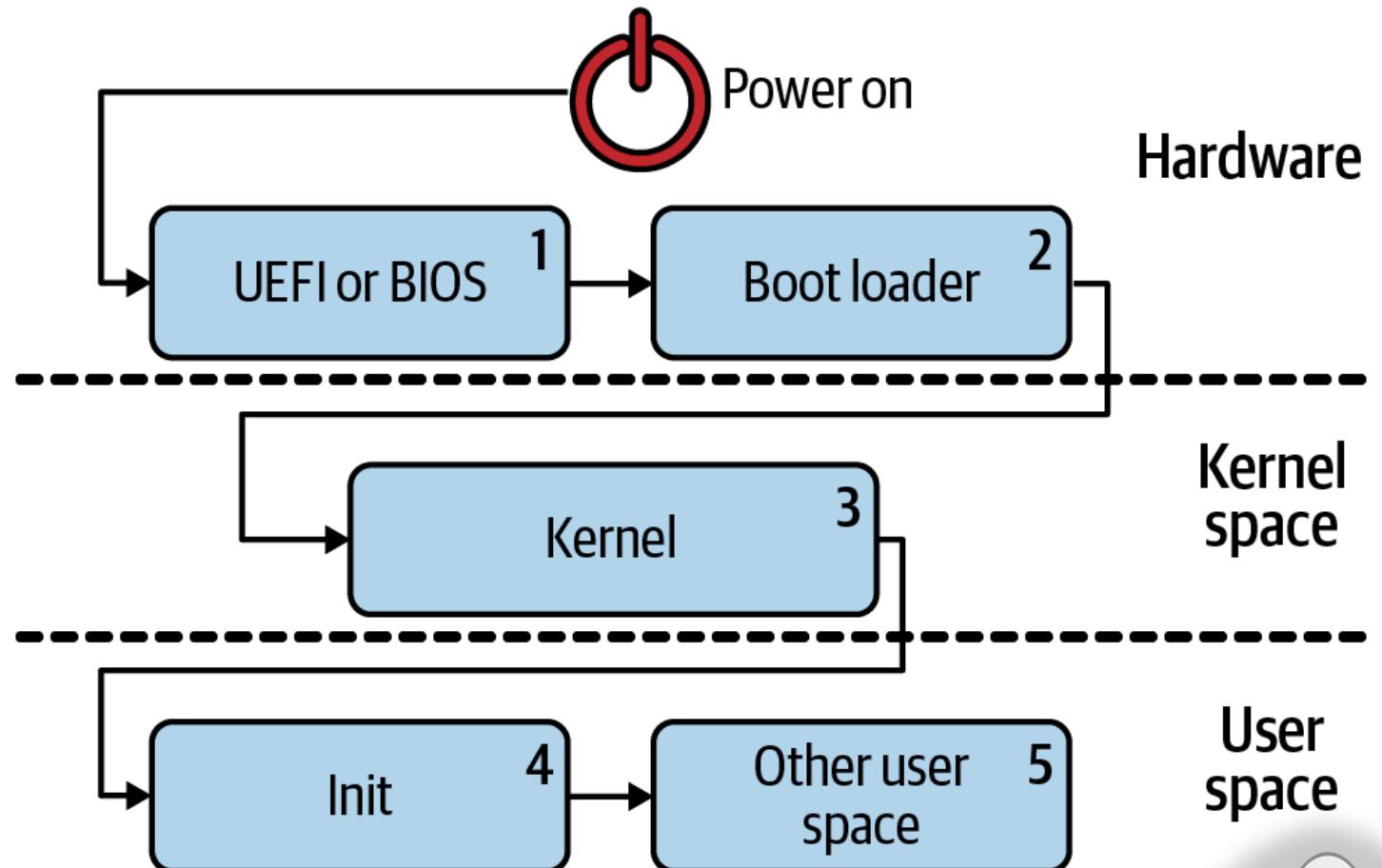


El proceso de arranque de Linux

- El proceso de arranque de Linux implica múltiples etapas y componentes de software.
- El proceso de arranque de Linux se deriva de los procesos de arranque de estilo BSD y Unix.
- Se utilizan diferentes variaciones y enfoques para cada una de las etapas y componentes del proceso de arranque de Linux.
- Los cargadores de arranque como GRUB, coreboot o Das U-Boot pueden ser utilizados para iniciar el proceso de arranque de Linux.
- Los scripts de inicio pueden ser tradicionales de estilo init o configuraciones modernas como systemd o Upstart.



El proceso de inicio de Linux



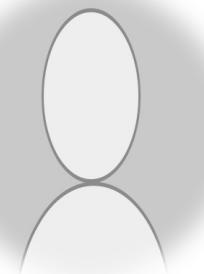
El proceso de inicio de Linux

1. UEFI o BIOS

En entornos modernos, UEFI define la configuración de arranque (almacenada en NVRAM) y el boot loader. En sistemas más antiguos, en este paso, después de completar la prueba automática de encendido, el BIOS inicializaría el hardware (administrando los puertos de I/O e interrupciones) y entregar el control al boot loader.

2. Boot Loader

Tiene un objetivo: arrancar el Kernel. Hay una variedad de opciones de cargador de arranque, tanto actuales (ej., GRUB 2, systemd-boot, SYSLINUX, rEFInd) como legadas (ej., LILO, GRUB 1).



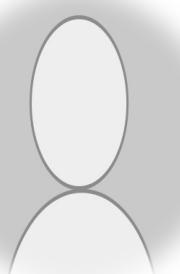
El proceso de inicio de Linux

3. Kernel

El Kernel generalmente se encuentra en el directorio /boot en forma comprimida. El primer paso es extraer y cargar el Kernel en la memoria principal. Después de la inicialización de sus subsistemas, filesystems y drivers, luego entrega el control al sistema init.

4. Init

Es responsable de lanzar demonios (servicios) en todo el sistema. Este proceso de inicio es la raíz de la jerarquía de procesos y tiene el ID de proceso (PID) 1. El proceso con PID 1 se ejecuta hasta que apaga el sistema. Además de ser responsable de lanzar otros demonios.

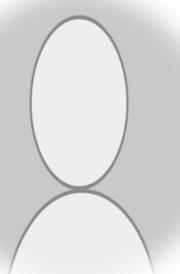


El proceso de inicio de Linux

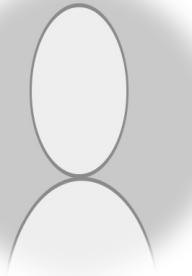
5. Other User Space

Por lo general, después de esto se lleva a cabo alguna otra inicialización a nivel de espacio de usuario, según el entorno:

- Por lo general, hay una inicialización de terminal, entorno y Shell
- Se inician el administrador de pantalla, el servidor gráfico y similares para entornos de escritorio con una GUI, teniendo en cuenta las preferencias y configuraciones del usuario.

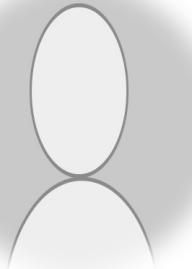


Init Systems



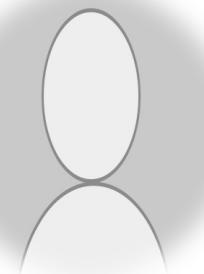
System V Init (init.d)

- Los programas de inicio de estilo System V (o SysV init) eran el sistema de inicio tradicional en Linux.
- SysV init se heredó de Unix y define diferentes niveles de ejecución que representan diferentes estados del sistema.
- La configuración de SysV init se guarda normalmente en /etc/init.d.
- El modo secuencial de inicio de los demonios y el manejo específico de la configuración por distribución hizo de SysV init una opción poco portátil.



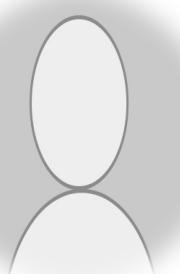
Systemd

- systemd proporciona un sistema y administrador de servicios que se ejecuta como PID 1 e inicia el resto del sistema
- Inicialmente fue creado como un remplazo de initd, pero hoy en día incluye funciones de logueo, configuraciones de red y sincronización de tiempo por red.
- Casi todas las distribuciones actuales de Linux utilizan systemd
 - Fedora desde mayo de 2011
 - openSUSE desde septiembre de 2012
 - CentOS desde abril de 2014
 - RHEL desde junio de 2014
 - SUSE Linux desde octubre de 2014
 - Debian desde abril de 2015 y Ubuntu desde abril de 2015.



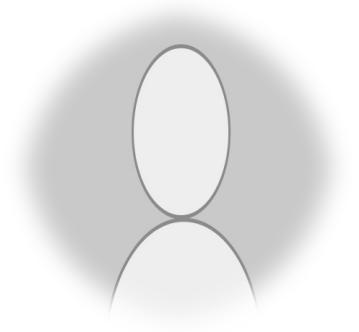
Mejoras de Systemd

- Proporcionar una forma uniforme de administrar el inicio en todas las distribuciones
- Implementación de una configuración de servicio más rápida y comprensible. Puede iniciar servicios en paralelo mientras todas sus dependencias estén presentes
- Ofreciendo una suite de administración moderna que incluye monitoreo, control de uso de recursos (a través de cgroups) y auditoría integrada



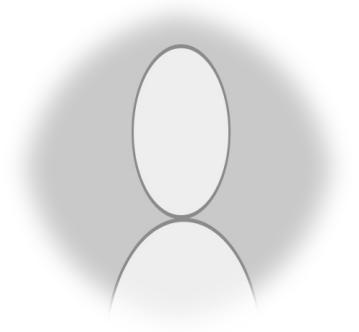
Units en Systemd

- La forma en que le dice a systemd qué ejecutar, cuándo ejecutar y cómo ejecutar es a través de units
- Un unit en systemd es una agrupación lógica con diferente semántica según su función y/o el recurso al que se dirige (target).
- Dependiendo del target destino estos son los units más comunes
 - Service units
 - Describir cómo gestionar un servicio o una aplicación.
 - Target units
 - Define dependencias
 - Mounts units
 - Define un punto de montaje
 - Timer units
 - Definir temporizadores para cronjobs y similares



Systemd Locations

- Para que systemd la conozca, un unit debe serializarse en un archivo. systemd busca archivos unitarios en varias ubicaciones. Las tres rutas de archivo más importantes son las siguientes:
 - /lib/systemd/system
 - Package-installed units
 - /etc/systemd/system
 - System admin-configured units
 - /run/systemd/system
 - Nonpersistent runtime modifications



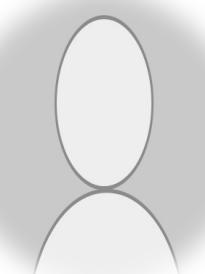
Manejo con Systemctl

- La herramienta que usa para interactuar con systemd para administrar servicios es systemctl
- Algunos comandos comunes pueden ser los siguientes:

Command	Use Case
systemctl enable XXXXX.service	Enable the service; ready to be started
systemctl daemon-reload	Reload all unit files and re-create entire dependency tree
systemctl start XXXXX.service	Start the service
systemctl stop XXXXX.service	Stop the service
systemctl restart XXXXX.service	Stop and then start the service

Otras tools de systemd

- **journalctl**
 - Proporcionar una ubicación centralizada para todos los logs registrados por los componentes systemd
- **bootctl**
 - Permite chequear el boot loader status y manejar los distintos boot loaders
- **timedatectl**
 - Permite configurar y ver información relacionada con la hora y la fecha.
- **coredumpctl**
 - Permite procesar volcados del Kernel guardados. Considere esta herramienta cuando esté solucionando problemas.



Actividad 4

Crear un systemd Unit de tipo servicio que realice lo siguiente.

- Ejecutar un script imprima un saludo y la fecha actual infinitamente con una pausa de un segundo.
- Habilitar el servicio para que se inicie con el sistema

Subir un readme file explicando el proceso de instalación del servicio y como poder chequear sus logs.

Detalles de entrega:

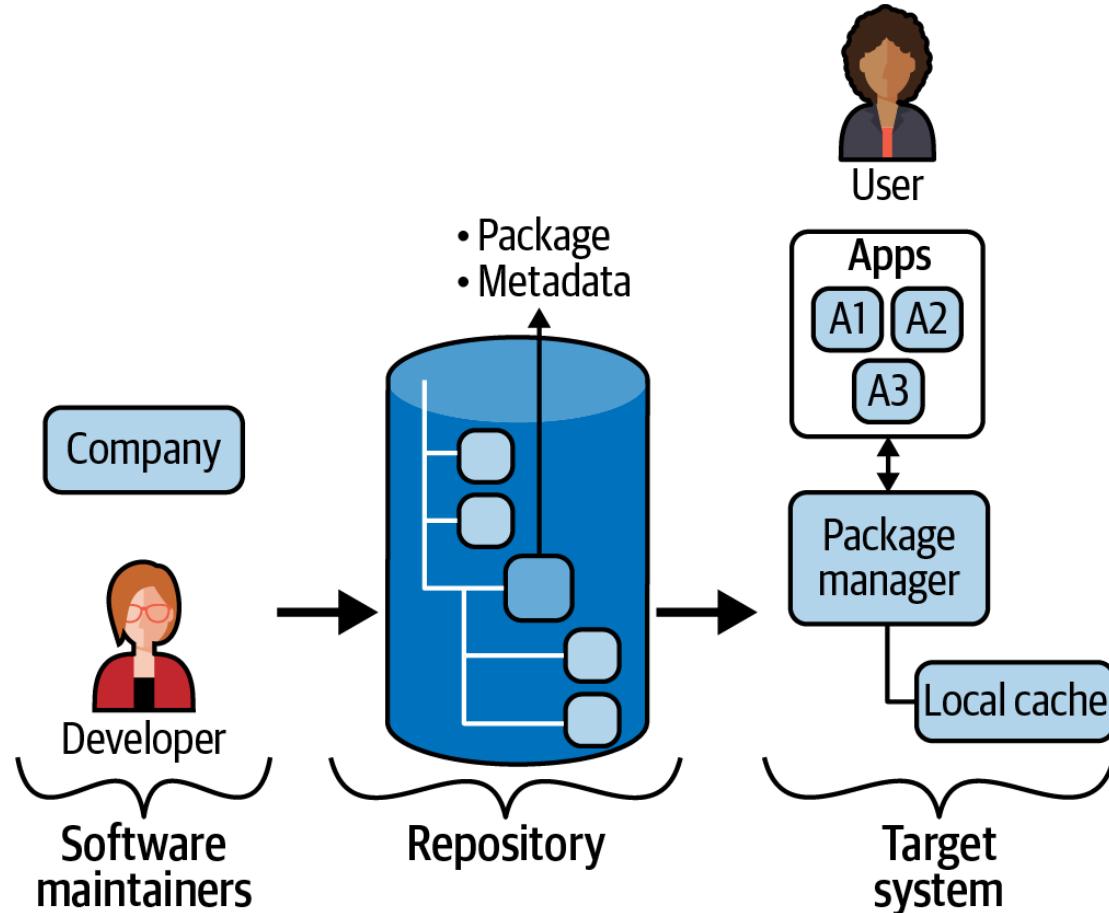
- Fecha de entrega: 17/08/24
- Entrega: Link al folder de GitHub
- Nombre del folder: **actividad4**

Linux Application Supply Chains

Linux Application Supply Chains

- Las cadenas de suministro se refieren a una red de individuos y organizaciones involucrados en el suministro de productos a los consumidores, como alimentos y combustibles.
- Estas también existen en el ámbito de las aplicaciones de software, donde los productos están compuestos por artefactos de software.
- Los consumidores en el contexto de las aplicaciones de software pueden referirse tanto a usuarios individuales como a herramientas que administran las aplicaciones en su nombre.
- Las cadenas de suministro son ubicuas y tienen un impacto en nuestra vida diaria, incluso si no pensamos en ellas con frecuencia.
- Comprender las cadenas de suministro es importante para garantizar la calidad y seguridad de las aplicaciones de software.

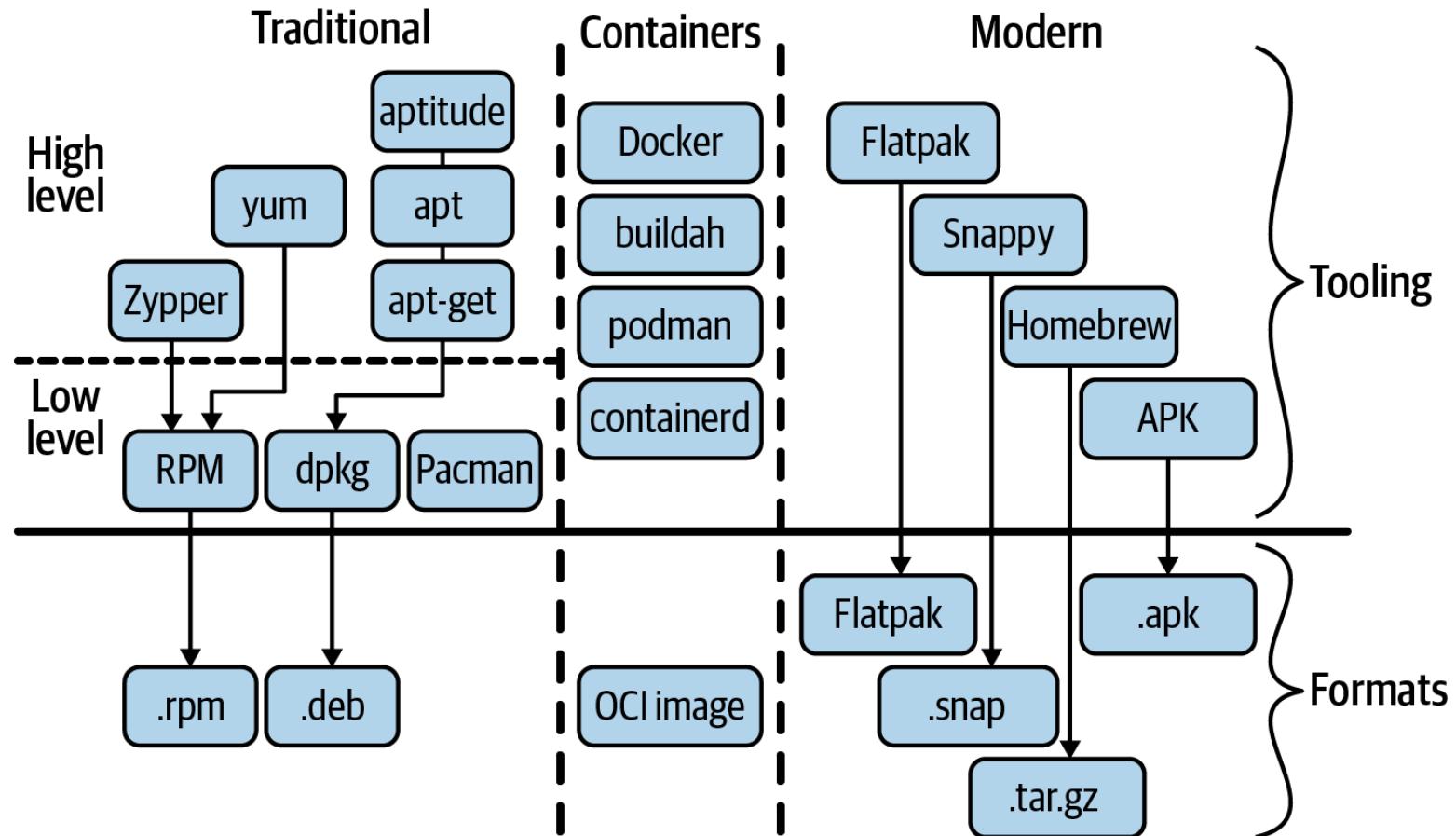
Principales actores en la cadena de suministro de apps



Principales actores

- Mantenedores de software
 - Estos incluyen desarrolladores individuales, proyectos de código abierto y empresas, como proveedores de software independientes (ISV), que producen artefactos de software y los publican, por ejemplo, como paquetes en un repositorio (repo).
- Repositorio
 - Esto enumera el paquete que contiene la totalidad o parte de una aplicación junto con los metadatos. El paquete generalmente captura las dependencias de una aplicación. Las dependencias son otros paquetes que una aplicación necesita para funcionar. Esto puede ser una biblioteca, algún tipo de exportadores o importadores, u otros programas de servicio. Mantener estas dependencias actualizadas es difícil.
- Herramientas (un administrador de paquetes)
 - En el lado del sistema de destino, esto puede buscar paquetes en el repositorio e instalar, actualizar y eliminar aplicaciones según las instrucciones del usuario humano. Tenga en cuenta que uno o más paquetes pueden representar la aplicación y sus dependencias.

El universo de gestión de dependencias y apps en Linux



Packages and Package Managers

Packages and Package Managers

- Estos generalmente provienen de dos principales Linux familias de distribución:
 - Red Hat (RHEL, Fedora, CentOS, etc.)
 - Sistemas basados en Debian (Debian, Ubuntu, etc.)
- Estos pueden ser instalados en escritorio, VMs, o servidores
- Conceptos claves
 - Paquetes

Técnicamente, un archivo que generalmente está comprimido y puede contener metadatos.
 - Administradores de paquetes

Se ocupa de esos paquetes en el sistema de destino, para instalar y mantener aplicaciones. Un administrador de paquetes generalmente interactúa con el repositorio en su nombre y mantiene un caché local de paquetes.

RPM Package Manager

- RPM (Red Hat Package Manager) es un sistema de gestión de paquetes utilizado por algunas distribuciones de Linux, como Red Hat Enterprise Linux y Fedora. RPM se utiliza para distribuir, instalar y actualizar software en sistemas Linux.
- El formato de archivo .rpm se usa en Linux Standard Base y puede contener archivos binarios o fuente.
- Algunos administradores de paquetes que usan RPM incluyen lo siguiente:
 - ***yum***
 - In Amazon Linux, CentOS, Fedora, and RHEL
 - ***DNF***
 - In CentOS, Fedora, and RHEL
 - ***Zypper***
 - In openSUSE and SUSE Linux Enterprise

Debian deb

- Los paquetes deb y el formato de archivo .deb tienen su origen en la distribución Debian de Linux.
- Los paquetes deb pueden contener archivos binarios o de código fuente y son utilizados por múltiples gestores de paquetes, tanto de bajo nivel como dpkg, como de alto nivel, como apt-get, apt y aptitude.
- Dado que Ubuntu se basa en Debian, los paquetes deb son ampliamente utilizados tanto en sistemas de escritorio como en servidores.
- La instalación de paquetes deb se realiza comúnmente a través del comando "apt" y sus diversas opciones, como apt-get o aptitude.

Language-Specific Package Managers

- C/C++
 - Have many different package managers, including Conan and vcpkg
- Go
 - Has package management built in (go get, go mod)
- Node.js
 - Has npm and others
- Java
 - Has maven and nuts and others
- Python
 - Has pip and PyPM
- Ruby
 - Has rubygems and Rails
- Rust
 - Has cargo

Contenedores

Contenedores

- Un contenedor es una unidad de software que encapsula código y todas sus dependencias, permitiendo la portabilidad y consistencia en diferentes entornos ejecución.
- Los casos de uso para contenedores van desde pruebas y desarrollo locales hasta trabajar con sistemas distribuidos, por ejemplo, microservicios en contenedores en Kubernetes.
- A bajo nivel los contenedores es un grupo de procesos de Linux que utiliza ***Linux namespaces***, ***cgroups*** y, opcionalmente, sistemas de archivos ***CoW***.

Historia de los contenedores

- Los contenedores no son nuevos en Linux, pero fue Docker quien lo volvió mainstream en 2014.
- Docker creó una interface que le permitió a los desarrolladores interactuar con Linux Namespaces y cgroups fácilmente, junto con un estándar de empaquetar y distribuir imágenes
- Intentos de introducir contenedores previos a Docker
 - [Linux-VServer \(2001\)](#)
 - [OpenVZ \(2005\)](#)
 - [LXC \(2008\)](#)
 - [Let Me Contain That for You \(lmctfy\) \(2013\)](#)

Open Container Initiative (OCI)

- Especificación de tiempo de ejecución
 - Define lo que un tiempo de ejecución necesita admitir, incluidas las operaciones y las fases del ciclo de vida.
- Especificación de formato de imagen
 - Define cómo se construyen las imágenes de contenedores, en función de los metadatos y las capas.
- Especificación de distribución
 - Define cómo se envían las imágenes de contenedores, de manera efectiva, la forma en que funcionan los repositorios en el contexto de los contenedores.

Linux Namespaces

- Los procesos en Linux tienen una vista global de los recursos. Para permitir a los procesos tener una vista local existen los Namespaces
- Los Namespaces se centran en la visibilidad, por lo que pueden utilizarse para aislar recursos.
- El aislamiento en este contexto se refiere principalmente a lo que un proceso ve, no necesariamente a un límite rígido (desde una perspectiva de seguridad).

Principales syscalls para crear Namespaces

- clone
 - Se utiliza para crear un proceso secundario que puede compartir partes de su contexto de ejecución con el proceso principal
- unshare
 - Se utiliza para eliminar un contexto de ejecución compartido de un proceso existente
- setns
 - Se utiliza para unir un proceso existente a un espacio de nombres existente

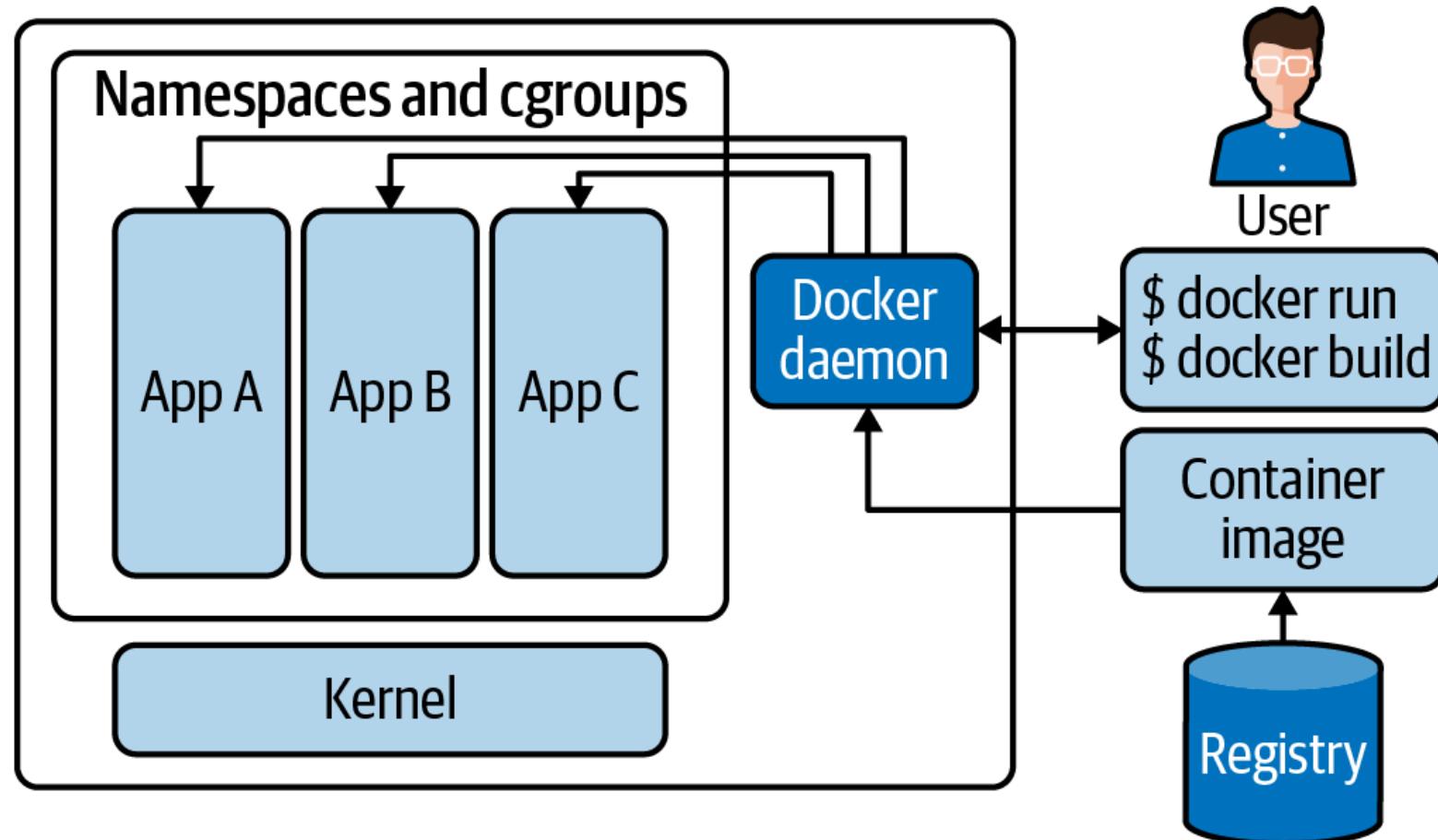
Linux cgroups

- Los cgroups (control groups) son un mecanismo en el kernel de Linux que permite limitar, priorizar y aislar los recursos de sistema, como CPU, memoria y dispositivos, entre diferentes procesos o grupos de procesos
- Linux Namespaces se enfoca en la visibilidad de recursos, mientras que los cgroups proporcionan la funcionalidad para organizar grupos de procesos.
- En este momento existen dos versiones de cgroups en el Kernel: cgroups v1 y v2. La idea es remplazar la v1 por la v2.

cgroups

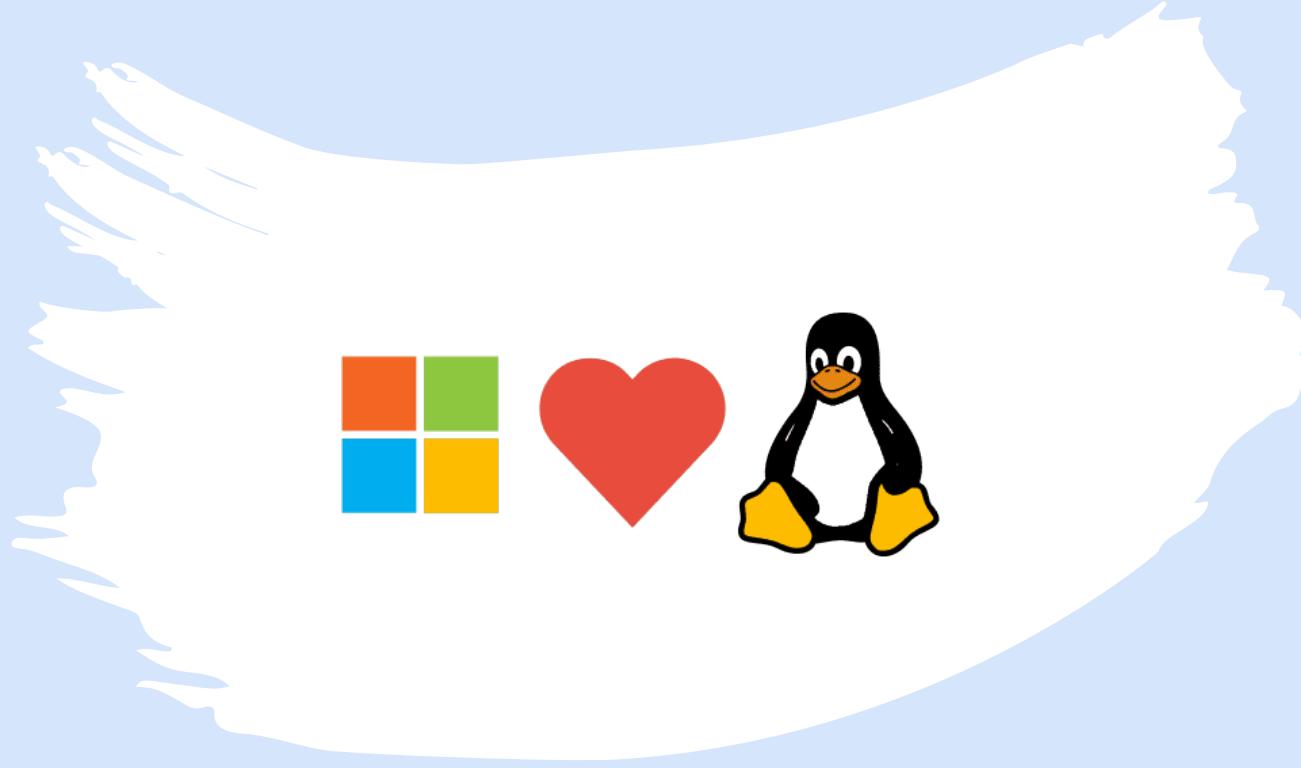
- cgroup v1
 - Con cgroup v1, la comunidad tenía un enfoque ad hoc, agregando nuevos cgroups y controladores según fuera necesario
- cgroup v2
 - Es una reescritura total de cgroups con las lecciones aprendidas de v1. es una mejora significativa sobre cgroups v1 en términos de simplicidad, control, aislamiento y eficiencia
 - cgroups v2 se está convirtiendo en el default, pero depende de la distribución de Linux que se esté utilizando y de la compatibilidad con las aplicaciones existentes. Distribuciones como Ubuntu 21.10, Arch y Fedora 31+ ya lo usan por defecto.

Arquitectura de Docker a alto nivel



Alternativas a Docker

- No tiene que usar Docker para trabajar con contenedores OCI; como alternativa, puede utilizar una combinación dirigida y patrocinada por Red Hat: **podman** y **buildah**.
- Estas herramientas sin daemon le permiten crear imágenes de contenedores OCI (buildah) y ejecutarlas (podman).
- **Containerd**
 - Un demonio que administra el ciclo de vida del contenedor OCI, desde la transferencia y el almacenamiento de imágenes hasta la supervisión del tiempo de ejecución del contenedor.



Muchas Gracias