RESUMEN EXAMEN FINAL DE TEORÍA LABORATORIO DE SISTEMAS:

TEMA 5: CONTROL DE VERSIONES (GIT)

- **[] 5.1 INTRODUCCIÓN A LOS SCV**
  - **5.1.1 REGISTRO DE CAMBIOS**
  - **5.1.2 CORRECCIÓN DE ERRORES Y PETICIÓN DE MEJORAS**
  - **5.1.3 CLONADO, FORK Y CONTRIBUCIONES**
  - **5.1.4 VERSIONES DE LANZAMIENTO (RELEASES)**
  - **5.1.5 SCV CENTRALIZADO Y DISTRIBUIDO**
  - **5.1.6 EJEMPLOS DE SCV**
- **5.2 SCV (GIT)**
  - **5.2.1 SCV**
  - **5.2.2 SCV DISTRIBUIDO**
  - **5.2.3 ¿QUÉ NO ES GIT?**
  - **5.2.4 ¿QUÉ COSAS PUEDO HACER?**
  - **5.2.5 TERMINOLOGÍA BÁSICA**
  - **5.2.6 FUNCIONAMIENTO INTERNO**
  - **5.2.7 CONFIGURACIÓN**
  - **5.2.8 FICHERO .GITIGNORE**
  - **5.2.9 GIT DIFF**
  - **5.2.10 TIPOS DE REPOSITORIOS**
- **5.3 GIT AVANZADO**
  - **5.3.1 CICLO DE CAMBIOS EN ARCHIVO DEL PROYECTO**
  - **5.3.2 PROPÓSITOS DE LAS RAMAS**
  - **5.3.3 CREACIÓN DE RAMAS**
  - **5.3.4 CAMBIOS Y COMMITS EN RAMAS**
  - **5.3.5 GESTIÓN DEL ÁREA DE TRABAJO**
  - **5.3.6 PULL REQUEST**
  - **5.3.7 ETIQUETAS**
  - **5.3.8 DIFF Y PATCH**
  - **5.3.9 MERGE Y CONFLICTOS**
- **5.4 LISTADO DE COMANDOS GIT**
  - **5.4.1 CONFIGURACIÓN DE GIT**
  - **5.4.2 TRABAJO DIARIO**
  - **5.4.3 COMENZANDO UN PROYECTO**
  - **5.4.4 MODELADO DE RAMAS EN GIT**
  - **5.4.5 REVISIÓN DEL TRABAJO REALIZADO**
  - **5.4.6 ETIQUETANDO COMMITS CONOCIDOS**
  - **5.4.7 REVIRTIENDO CAMBIOS**
  - **5.4.8 SINCRONIZANDO REPOSITORIOS**

[] TEMA 6: ARRANQUE DEL SISTEMA

- **[] 6.1 ARRANQUE DEL SISTEMA**
  - **6.1.1 SECUENCIA DE ARRANQUE EN UBUNTU Y LINUX**
  - **6.1.2 FIRMWARE**
  - **6.1.3 UEFI**
  - **6.1.4 CARGADOR**
  - **6.1.5 GRUB**
  - **6.1.6 LINUX SETUP**
  - **6.1.7 KERNEL**
  - **6.1.8 INITRD**
  - **6.1.9 INIT**
  - **6.1.10 SYSTEMD**
  - **6.1.11 COMANDOS DE SYSTEMD**
  - **6.1.12 RUNLEVELS**
  - **6.1.13 ENTORNO GRÁFICO**

TEMA 7: FICHEROS Y COMANDOS AVANZADOS

7.1 FICHEROS Y COMANDOS AVANZADOS

- **7.1.1 METADATOS E I-NODOS**
- **7.1.2 ENLACES DUROS**
- **7.1.3 TIPOS DE FICHEROS**
- **7.1.4 ENLACES BLANDOS**
- **7.1.5 PIPES CON NOMBRE**
- **7.1.6 DISPOSITIVOS**
- **7.1.7 EL COMANDO DD**
- **7.1.8 EL COMANDO RSYNC**
- **7.1.9 INSPECCIÓN DEL SISTEMA**
- **7.1.10 COMANDOS DE INSPECCIÓN DEL SISTEMA**
- **7.1.11 USUARIOS**
- **7.1.12 LOGS**
- **7.1.13 TRABAJO EN REMOTO**

TEMA 8: COMPILACIÓN, CARGA Y DEPURACIÓN DE PROGRAMAS

8.1 COMPILACIÓN, CARGA Y DEPURACIÓN

- **8.1.1 ELF**
- **8.1.2 COMPILACIÓN**
- **8.1.3 ENLAZADO**
- **8.1.4 LIBRERÍAS**
- **8.1.5 DEPURACIÓN**
- **8.1.6 PROFILING**
- **8.1.7 ANÁLISIS ESTÁTICO Y DINÁMICO**

TEMA 9: CONTENEDORES VIRTUALES

9.1 VIRTUALIZACIÓN (CONTENEDORES)

- **9.1.1 CONTENEDORES**
- **9.1.2 TIPOS DE DOCKER SEGÚN SU UBICACIÓN**
- **9.1.3 EJECUCIÓN DE CONTENEDORES SEGÚN SU UBICACIÓN**
- **9.1.4 CLASIFICACIÓN DE DOCKERS**
- **9.1.5 OTROS TIPOS DE CONTENEDORES**

[] TEMA 1: INTRODUCCIÓN (SHELL DE UNIX-LINUX)

[] 1.1 SHELL BÁSICO

} 1.1.1 USO BÁSICO DE LA SHELL DE UNIX/LINUX
Unix: Familia de sistemas operativos a la que pertenece Linux.
Shell: Programa que nos permite manejar nuestro sistema usando solo teclado y pantalla en modo texto, sin gráficos ni ratón.

} 1.1.2 TERMINAL
Terminal: Combinación de teclado y pantalla sin gráficos (trabajo local o remoto).

} 1.1.3 SESIÓN
Sesión: Intercambio de información entre el usuario y el ordenador (en modo texto se escribe al terminal).

} 1.1.4 DIRECTORIOS
Home: Sitio reservado para el usuario donde se guarda su trabajo, representado por la virgulilla (~).

} 1.1.5 OPCIONES
Estructura de comandos: rm (orden), -r (opción), probando (argumento).

} 1.1.6 PROMPT
Prompt: Línea de texto que aparece en el terminal cuando la shell se prepara para ejecutar una orden.
Prompt (estructura): jperez (nombre de usuario), @, f-l-vm01 (nombre del host), :, directorio actual (~), $.

} 1.1.7 PATH
Path: Texto de forma compacta que especifica dónde está un fichero (~/fpi/practica01/holamundo.pas).

} 1.1.8 ÓRDENES BÁSICAS
ls: Lista el contenido de un directorio (ficheros y subdirectorios), ls -l (salida larga, muestra más atributos).
cd: Cambia el directorio actual para entrar o salir de él, cd (home), cd .. (sale del directorio actual).
mkdir: Crea directorios (no es lo mismo mkdir fpi que mkdir ~/fpi).
rm: Borra uno o más ficheros, rm -r (borra ficheros y también directorios recursivamente).
tree: Muestra la estructura en forma de árbol de todo el contenido del directorio actual.
Visualización de salidas largas: Barra de desplazamiento vertical o añadir la barra vertical y el comando less.
exit: Finaliza la shell actual y cierra la sesión.

[] 1.2 SHELL CLI (INTERFAZ DE LÍNEA DE COMANDOS)

} 1.2.1 SHELL CLI
Shell: Programa que actúa como interfaz entre usuario y sistema operativo y que le permite introducir órdenes para que las ejecute.
Trabajo con prompt: Escribir órdenes en la línea de comandos para ser interpretadas y ejecutadas por el SO.
Bucle: Cursor -> Introducir comando -> Interpretar comando -> Ejecutar comando -> Salir de comando -> …

} 1.2.2 SHELL SCRIPTING (BASH)
Trabajo con scripts: Fichero con órdenes del sistema operativo como contenido similares a las de Pascal o C.
which <nombre programa>: Comprobar instalación de licencias de la shell.

} 1.2.3 ÓRDENES EXTERNAS E INTERNAS
Órdenes externas: Se encuentran en un fichero ejecutable dentro de los directorios listados en $PATH.
Órdenes internas: Se ejecutan sin necesidad de buscar el ejecutable ya que tienen incluida su funcionalidad en el propio código (cd, exit, history, fg, bg, jobs).

} 1.2.4 FOREGROUND VS BACKGROUND

Foreground (ejecución de órdenes externas interactivas), background (programa no interactivo, &).
Ejemplo de background: xlogo & -> [número de tarea / job] identificador de proceso (PID).
bg: Pasa a segundo plano un proceso suspendido (Ctrl + z).
fg: Trae a primer plano un proceso que se encontraba en segundo plano.
jobs: Lista los procesos que se están ejecutando en segundo plano, cuando uno finaliza la shell lo notifica.
kill <PID> <%número de tarea / job>: Termina la ejecución de un proceso en background (o SIGKILL).

} 1.2.5 VARIABLES Y ARITMÉTICA

Operaciones aritméticas: ancho=24 // ancho=$(($ancho+1)) // echo $ancho -> 25.
export: Convierte una variable normal (sólo en shell) en una variable de entorno (cualquier programa de shell).
printenv: Lista las variables de entorno.

} 1.2.6 REDIRECCIONES

Descriptores de ficheros: 0 (entrada estándar), 1 (salida estándar o >, con >> añadimos la salida a un fichero existente), 2 (salida de error o 2>), >& (redirige al fichero tanto mensajes de error como los de salida normal).

} 1.2.7 CAUCES (PIPES)

Cauces (pipes): Encadena órdenes con el operador |, donde la salida de una es la entrada de la siguiente.

} 1.2.8 LISTAS DE ÓRDENES

Listas de órdenes: ; (ejecución secuencial, devuelve $?), & (ejecución en segundo plano, devuelve 0), && (orden2 si orden1 devuelve 0), || (orden2 si orden1 devuelve !=0).

ls /bin/bash && echo Existe ; echo $? => Retorna /bin/bash, Existe y 0.
ls /bin/noexiste && echo Existe ; echo $? => Retorna error y 2.
ls /bin/bash || echo No existe ; echo $? => Retorna /bin/bash y 0.
ls /bin/noexiste || echo No existe ; echo $? => Retorna error, No existe y 0.

} 1.2.9 EXPANSIÓN DE COMODINES Y DE ÓRDENES

Expansión de comodines: * (cualquier cadena de caracteres), ? (cualquier carácter), [conjunto] (cualquier carácter dentro del conjunto).
Expansión de órdenes (ejemplo): num=$(ls a* | wc -w) // echo $num -> 13.

} 1.2.10 ÓRDENES ÚTILES

grep: Busca en la entrada líneas que coincidan con el patrón dado.
wc: Escribe el número de saltos de línea, palabras o caracteres de un fichero.
sort: Escribe la concatenación ordenada del fichero a la salida estándar.
cut: Escribe en la salida estándar partes seleccionadas de cada línea de sus ficheros de entrada.

[] TEMA 2: SHELL SCRIPTING

[] 2.1 SHELL SCRIPTING

} 2.1.1 SCRIPTS DE SHELL

Scripts de shell: Programas interpretados mediante el mecanismo hash bang.
!#/bin/sh: Programa interpretado y que el comando que lo interpreta es la shell en la ruta /bin/sh, si se añade el parámetro -x depura el programa y hace que la shell escriba el comando antes de ejecutarlo.
POSIX: Características comunes a todas las shells para ser portables entre diferentes sistemas.

} 2.1.2 EJECUCIÓN DE SCRIPTS DE SHELL

$?: Muestra el estado con el que acabó la ejecución del último comando o pipeline.

} 2.1.3 PARÁMETROS POSICIONALES

Parámetros de entrada: $0 (nombre del script), $# (número de parámetros), $* (parámetros posicionales), "$*" (igual que $* pero en una sola cadena), $@ (igual que $* pero separados en varias cadenas), "$@" ("$1" "$2" …), shift (desplaza los parámetros, $3 -> $2 y actualiza $#).

```
#!/bin/sh                                   $ ./param.sh -a -b -c fich
echo \$0 es $0                              $0 es ./param.sh
echo \$\# es $#                             $# es 4
echo \$\* es $*                             $* es -a -b -c fich
echo $1 $2 $3 $4              =>            -a -b -c fich
echo \$\@ es $@                             $@ es -a -b -c fich
shift                                       -b -c fich
shift                                       -c fich
echo $1 $2 $3 $4 $#                         -c fich 2
```

} 2.1.4 CONTROL DE FLUJO Y OTROS ELEMENTOS DE SHELL SCRIPTING

read: Lee una línea de la entrada estándar y la guarda en una variable que se le pasa como argumento.

```
echo 'a b
        c d' > /tmp/e

while read line
do
            echo $line
done < /tmp/e
Salida: a b \n c d

for x in 'cat /tmp/e'
do
            echo $x
done
Salida: cat /tmp/e
```

Variable IFS: Contiene los caracteres reconocidos como separadores entre campos (\t, \n y ' ').

```
export IFS=-
for i in $(echo uno dos tres); do echo $i; done
Salida: uno dos tres
```

alias <nombre>=<comandos>: Define una etiqueta para invocar un comando o conjunto de comandos de forma directa (alias muestra los ya definidos y unalias elimina uno previamente definido).
Test (ficheros): -f fich (existe fichero), -d dir (existe directorio).
Test (cadenas): -n str (longitud != 0), -z str (longitud = 0), str1 = str2 (iguales), str1 != str2 (diferentes), str (cadena no nula).
Test (enteros): int1 -eq int2 (iguales), int1 -ne int2 (diferentes), int1 -gt int2 (mayor que), int1 -ge int2 (mayor o igual que), int1 -lt int2 (menor que), int1 -le int2 (menor o igual que).
Sintaxis para test: [$a -eq $b] == test $a -eq $b.
bc: Permite realizar operaciones básicas que involucran números enteros.
du -a // find .: Recorrer un árbol de ficheros.
join: Calcula la intersección de valores presentes en dos columnas previamente ordenadas.

```
$ echo '
a uno
b dos
c tres' > a.txt
```

```
$ echo '
a cuatro
b cinco
c seis' > b.txt
$ join a.txt b.txt
a uno cuatro
b dos cinco
c tres seis
```

xargs: Usa lo que nos viene por la entrada estándar como argumento de entrada al ejecutar otro comando.

```
$ echo a b c | xargs ls -l
-rw-rw-r-- 1 jfelipe jfelipe 2 mar1 16:03 a
-rw-rw-r-- 1 jfelipe jfelipe 2 mar1 16:03 b
-rw-rw-r-- 1 jfelipe jfelipe 2 mar1 16:03 c
```

```
$ echo uno dos tres | cut -d' ' -f 1,3
uno tres
$ ps | sed 3q > a
$ seq 1 3 > b
$ paste a b
PID        TTYTIME        CMD1
8462       pts/400:00:00  bash2
11357      pts/400:00:00  ps3
$ paste b a
1          PID TTY          TIME CMD
2          8462 pts/4       00:00:00 bash
3          11357 pts/4      00:00:00 ps
```

[] 2.2 FILTROS Y EXPRESIONES REGULARES

} 2.2.1 FILTROS ÚTILES

sort: Ordena las líneas de diversas formas (lista de columnas -kcol1,col2, separador -t\sep, intervalo de campos, estabilidad -s).

```
$ cat x.txt
1-2-4
2-3-3
2-2-1
2-1-4
```

```
$ sort -k2,2 -t\- x.txt
2-1-4
2-2-1
1-2-4
2-3-3
```

```
$ sort -k1,2 -t\- x.txt
1-2-4
2-1-4
2-2-1
2-3-3
```

Comandos de filtro: uniq (elimina líneas contiguas repetidas), head (primeras líneas), tail (últimas líneas).

Comparación de ficheros: diff (ficheros de texto línea a línea), cmp (ficheros binarios byte a byte).
tr: Traduce un conjunto de caracteres, -d borra los caracteres del conjunto pasado como argumento.

} 2.2.2 EXPRESIONES REGULARES

Elementos expresiones regulares: . (cualquier carácter), [conjunto] (cualquier carácter dentro del conjunto), [^conjunto] (cualquier carácter que no esté dentro del conjunto), ^ (principio de línea), $ (final de línea), exp* (cero o más veces, a* -> ‘’, a, aa, etc), exp+ (una o más veces, ba+ -> baa, baaa, etc), exp? (cero o una vez), (exp) (agrupa exp), exp | exp (or lógico), \ (carácter de escape, el símbolo pierde su significado especial).
egrep: Filtra líneas usando expresiones regulares.
egrep (opciones): -v (líneas que no corresponden a exp), -n (número de línea de coincidencias con exp), -e (usa el siguiente argumento como una expresión), -q (no devuelve nada pero sí status de salida), -i (insensible a mayúsculas y minúsculas).

} 2.2.3 SED

sed (opciones): -E (uso de exp extendidas), q (sale del programa), d (borra la línea), p (imprime la línea), r (lee e inserta un fichero), s (sustituye).
sed (direcciones): número (actúa en la línea número), /exp/ (líneas que coinciden con exp), $ (última línea).
sed (intervalos): número,número (actúa en el intervalo número,número), número,$ (desde la línea número hasta la última línea), número,/exp/ (desde la línea número hasta la primera línea que encaja con exp).

sed -E '3,6d': Borra las líneas de la 3 a la 6.
sed -E -n ’/BEGIN|begin/,/END|end/p’: Imprime las líneas entre las que coinciden con esas regex.
sed -E '3q': Imprime las tres primeras líneas.
sed -E -n ’13,$p’: Imprime desde la línea 13 a la última.
sed -E ’/[Hh]ola/d’: Borra las líneas que contengan Hola u hola.
sed -E ‘s/exp/sustitución/’: Busca la primera cadena que coincide con exp y la sustituye.
sed -E ‘s/exp/sustitución/g’: Busca todas las cadenas que coinciden con exp y las sustituye.
sed -E ‘s/(exp1)exp2(exp3)/\1sustitución\2/g’: Las cadenas corresponden con su backreference.
sed -E ’s/[0-9]/X/: El primer dígito de la línea se sustituye por una ’X’.
sed -E ’s/[0-9]/X/g’: Todos los dígitos de la línea se sustituyen por una ’X’.
echo ‘Alberto 9’ | sed -E ’s/^([A-Za-z]+)[ ]+([0-9]+)/NOMBRE:\1 NOTA:\2/g’: NOMBRE:Alberto NOTA:9.
sed -E -n '/^(.)(.)(.)\3\2\1$/p' /usr/share/dict/words: Busca palíndromos de 6 letras en el fichero /usr/share/dict/words.
echo 'James Bond' | sed -E 's/(.*) (.*)/My name is \2, \1 \2./'.: My name is Bond, James Bond.

} 2.2.4 AWK

awk print: Imprime los operadores, si se separan con , (espacio), al final imprime salto de línea.
awk printf(): Imprime permitiendo controlar el formato de la salida.
Variables: $0 (línea procesada), $1, $2, … (número de columna), NR (número de línea a procesar), FILENAME (nombre del archivo a procesar), NF (número de campos del registro procesado), var=x o -v (declaración y paso de variables al programa), patrón {programa} (procesa líneas correspondientes al patrón).
BEGIN: Se ejecuta una sola vez al principio (encabezados, inicialización de variables, bloque opcional).
Cuerpo: Se ejecuta una vez por cada línea de entrada (no se marca con ninguna palabra reservada especial).
END: Solo se ejecuta una vez al final (final de texto, limpieza de variables, uno o varios comandos AWK, bloque opcional).

} 2.2.5 ARRAYS ASOCIATIVOS Y OPERADORES

Arrays asociativos (sintaxis): arrayname[string] = value.
Operadores unarios: +variable (retorna la propia variable), -variable (cambia de signo el valor de la variable), ++variable (incremento y accedo al valor), –variable (decremento y accedo al valor), variable++ (accedo al valor e incremento), variable– (accedo al valor y decremento).
Operadores aritméticos: num1+num2 (suma), num1-num2 (resta), num1*num2 (multiplicación), num1/num2 (división), num1%num2 (módulo).
Operadores con cadenas: str1 str2 (concatena los valores de str1 y str2).

Operadores de asignación: = (asignación), += (suma y asignación), -= (resta y asignación), *= (producto y asignación), /= (división y asignación), %= (módulo de división y asignación).
Operadores de comparación: > (mayor que), >= (mayor o igual que), < (menor que), <= (menor o igual que), == (igual que), != (distinto que), && (and lógico), || (or lógico).
Operadores con regex: ~regex (coincidencia parcial), !~regex (si no coincide con la regex).

ps aux | awk '{dups[$1]++} END{for (user in dups) {print user, dups[user]}}': Procesos de cada usuario.

[] 2.3 PYTHON SCRIPTING

} 2.3.1 ESTRUCTURA DE UN SCRIPT EN PYTHON

Formas de ejecución: Importarlo como módulo (python3 -m fich.py) o ejecutarlo como script (if __name__ == "__main__" comprueba si se está ejecutando en el top-level code environment o no).
sys.exit(code): Finaliza el script devolviendo un código de status.

} 2.3.2 BIBLIOTECAS ESTÁNDAR EN PYTHON

```
def echo(text: str, repetitions: int = 3) -> str:
  """Imitate a real-world echo."""
  echoed_text = ""
  for i in range(repetitions, 0, -1):
          echoed_text += f"{text[-i:]}\n"
  return f"{echoed_text.lower()}."
if __name__ == "__main__":
  text = input("Yell something at a mountain: ")
  print(echo(text))
```

Salida: echo cho ho o .

sys.argv: Lee argumentos de la línea de comandos.
Módulo argparse: Parsea opciones, argumentos y subcomandos pasamos al invocar a nuestro script para ejecutarlo.

```
#!/usr/bin/python3
import argparse
parser = argparse.ArgumentParser(description='Process some integers.')
parser.add_argument('integers', metavar='N', type=int, nargs='+',
                    help='an integer for the accumulator')
parser.add_argument('--sum', dest='accumulate', action='store_const',
                    const=sum, default=max,
                    help='sum the integers (default: find the max)')
args = parser.parse_args()
print(args.accumulate(args.integers))
```

Salida: -h (ayuda), 2 3 4 (imprime el máximo -> 4), –sum 2 3 4 (imprime la suma -> 9).

Objeto ArgumentParser: Lee e interpreta los argumentos de entrada que se le pasan al programa.
Parámetros: usage (describe el uso del programa), description (texto de ayuda para los argumentos), epilog (texto de ayuda después de la lista de argumentos), add_help (añade la opción -h o –help al parseador), add_argument (permite añadir nuevos argumentos a la lista reconocida por el parser).
Otros argumentos: name/flags (primer argumento obligatorio), action (tipo de acción del argumento, algunos de sus posibles valores son store, store_const, store_true, store_false,append), const (guarda un valor constante que se añade a los atributos), default (valor asignado al argumento), type (tipo del argumento diferente de string), dest (cambio de nombre de la variable destino), help (descripción breve del argumento), metavar (identificador del valor pasado como argumento).
parse_args: Activa el parseo de los argumentos y opciones de entrada introducidos.

} 2.3.3 AUTOMATIZACIÓN DE TAREAS

Módulo re: Incorpora soporte para usar expresiones regulares en Python.
Módulo os: Ofrece una interfaz para acceder a los servicios del sistema operativo.
Módulo time: Devuelve la fecha y hora actuales y programa la ejecución de tareas.

[] TEMA 3: INSTALACIÓN Y GESTIÓN DE PAQUETES

[] 3.1 INSTALACIÓN Y GESTIÓN DE PAQUETES

} 3.1.1 INSTALACIÓN DE LINUX

Distribución: Colección concreta de software de área de usuario y un núcleo (kernel) del sistema operativo.
Elementos: Instalador, organización de directorios, ficheros de configuración, herramientas, software propietario, esquema de versiones, compatibilidad con la arquitectura de la CPU, etc.
Ejemplos de gestión de paquetes: apt (Debian .deb), RPM (RedHat .rpm), pacman (ArchLinux .tar).
Datos del sistema: /boot (ficheros de arranque), /proc y /sys (interfaz para interaccionar con el núcleo).
Configuración: /etc (ficheros de configuración).
Dispositivos: /dev (dispositivos físicos), /media y /mnt (puntos de montaje).
Bibliotecas: /lib y /usr/lib (bibliotecas de los ejecutables).
Usuarios: /home (carpetas de usuario).
Programas ejecutables: /bin, /usr/bin, /sbin (ejecutables del sistema), /opt (aplicaciones de terceros).
Datos temporales y en tiempo de ejecución: /var (datos generados en ejecución), /tmp (ficheros temporales, se borran en cada reinicio del sistema).
Almacenamiento persistente: Mantiene los datos almacenados aunque se apague la alimentación.
Almacenamiento secundario: Disco duro magnético, discos SSD, pendrives, discos ópticos.
Partición: Zona del disco independiente (/dev/sda -> /dev/sda1 y /dev/sda2).
fdisk: Manipula las particiones de un disco (parted y gparted lo hacen de forma más cómoda).
Esquema de partición y cargador: Los más utilizados son UEFI GPT y ESP FAT, respectivamente.
swap: Define un área de intercambio en el que el sistema vuelca datos de la memoria a disco si empieza a escasear espacio de la RAM.
Punto de montaje: Lugar del árbol de directorios donde está montado un disco junto con su contenido, se pueden consultar con las instrucciones mount o df.
Montaje de un sistema de ficheros (pasos): Crear directorio (mkdir /var), visibilidad del sistema de ficheros (mount -t ext4 -o rw /dev/sda3 /var), montaje automático y permanente (/etc/fstab), sistema de ficheros invisible (unmount /var).

} 3.1.2 GESTIÓN DE PAQUETES SOFTWARE

Distribución Ubuntu: Heredera de Debian, Desktop (usuarios finales con entorno de escritorio preinstalado) y Server (servidores de alto rendimiento y sin entorno de escritorio preinstalado), versiones de 6 meses y LTS.
lsb release -a: Información de la versión de Ubuntu.
uname -a: Información de la versión del Kernel del Linux.
apt update // apt upgrade: Mantiene la instalación al día.
do-release-upgrade -p: Actualiza la distribución a una nueva versión.
ubuntu-drivers devices: Paquetes recomendados de drivers para instalar.
apt download xia: Descarga de ficheros de prueba dentro de un directorio (ar x f.deb).
dpkg: Lugar donde se instalan y borran los ficheros y permite manipular ficheros .deb individuales.
Usos de dpkg: dpkg -i xia 2.2-3 all.deb, dpkg -r xia.

} 3.1.3 ADVANCED PACKAGE TOOL (APT)

apt: Herramienta más sencilla y potente, usa repositorios, direcciones en /etc/apt/sources.list(.d) (contienen binarios deb, código fuente deb-src y remotos https), se pueden añadir repositorios alternativos (PPAs, add-apt-repository ppa:libreoffice/ppa).
Secciones de repositorios: Main (S, F), Restricted (S, NF), Universe (NS, F), Multiverse (NS, NF).

hold: El paquete queda retenido al solicitar la instalación de un paquete (dependencia de otro no disponible, retención a mano, o dependencia de otro pendiente de instalación -> apt-get install nombre-paquete).
aptitude remove <paquete>: Desinstala un paquete resolviendo conflictos.
aptitude -purge remove <paquete>: Borra todos los archivos de configuración relacionados con el paquete.
aptitude dist-upgrade: Actualiza todos los paquetes, pasando si es necesario a una versión más reciente de la distribución.
aptitude clean: Borra la caché local eliminando todos los archivos .deb descargados.
aptitude search <regexp>: Busca una regexp en los campos nombre o descripción de los paquetes.
dpkg -l <cadena>: Muestra el estado de instalación del paquete.
aptitude show <paquete>: Muestra una descripción completa del paquete.
dpkg-reconfigure <paquete>: Dispara de nuevo el proceso de configuración del paquete.
apt autoremove: Elimina los paquetes que el usuario no ha instalado correctamente y no necesarios.
apt clean: Libera espacio de la caché de APT (/var/cache/apt/archives/*).
dpkg -S fichero: Muestra a qué paquete ya instalado pertenece a un fichero.
apt-file search fichero: busca qué paquete del repositorio contiene un fichero.

[] TEMA 4: GESTIÓN DE CUENTAS, USUARIOS Y GRUPOS

[] 4.1 CUENTAS, USUARIOS Y GRUPOS

} 4.1.1 USUARIOS Y GRUPOS

Sistemas Linux: Multiusuario (varios usuarios en el mismo sistema de forma simultánea), usuarios y grupos (comandos whoami e id).
Usuario: Permiten identificar quién ha lanzado un proceso y otorga permisos a los procesos.
Login: Proceso de autenticar a un usuario para que acceda a su cuenta, algunos usuarios no pueden hacer login (pseudo-usuarios) ya que ejecutan procesos del sistema (demonios) o por algún tipo de escalado de privilegios (sudo).
Identificadores: Usuario (uid) y grupo (gid), en el espacio de usuario se trabaja con nombres y se traducen mediante ficheros que mantienen datos de correspondencia entre ambos.
/etc/passwd: Fichero de texto con la traducción entre login-names y uids de usuarios.
Estructura: username:x:uid:gid:nombre en formato legible:ruta $HOME:ruta intérprete de línea de comandos.
/etc/shadow: Fichero donde se almacenan las contraseñas que no todos pueden leer.
Superusuario root: Máximos privilegios (UID,GID=0), permisos totales para cualquier acción.

} 4.1.2 GESTIÓN DE USUARIOS

Creación y eliminación de usuarios: adduser username, passwd username, deluser username.
Credenciales de un proceso: PID (identificador de proceso), UID (identificador de usuario), GID (identificador de usuario), EUID (identificador de usuario efectivo), EGID (identificador de grupo efectivo).
/bin/id: Devuelve tu UID y GID.
/bin/su: Ejecuta una shell con otro UID, se ejecuta como superusuario por omisión (UID=0).
/usr/bin/sudo: Ejecuta un comando con otro UID pero con la contraseña del usuario actual (/etc/sudoers).
pepe ALL = (root, bin : operator, system) ALL: Cualquier máquina, UID de root y bin, GID de operator y system, puede ejecutar cualquier comando.
paco laptop1 = NOPASSWD: /bin/kill, PASSWD: /bin/ls, /usr/bin/lprm: paco puede en la máquina laptop1, UID de root, kill sin contraseña, ls y lprm con contraseña.
Permisos POSIX: Metadatos de ficheros y directorios que indican las acciones que pueden realizar.
ls -l: Muestra información sobre ficheros y directorios en formato largo, ls -ld muestra solo directorios.
Estructura: Permisos, hard links, usuario propietario, grupo propietario, tamaño en bytes, fecha y hora de la última modificación, nombre del elemento.
Permisos POSIX (elementos): - (fichero regular), d (directorio), l (enlace simbólico), p (pipe con nombre), s (socket), c (dispositivo orientado a carácter), b (dispositivo orientado a bloque).
Permisos POSIX (estructura -> -rwxrw-r–): Caracter 1 (tipo de elemento), caracteres 2-4 (permisos del usuario propietario), caracteres 5-7 (permisos del resto de usuarios del mismo grupo), caracteres 8-10 (permisos del resto de usuarios del mismo sistema).

Permisos POSIX (tipos): Modo de acceso (conjunto de permisos), r (lectura), w (escritura), x (ejecución).
sticky bit (+t): Impide borrar una entrada si no eres dueño del directorio que representa la entrada o root.
setuid/setgid bit (+s): El proceso a ejecutar adoptará el UID/GID efectivo del dueño/grupo del fichero.
chmod: Cambia los permisos de un fichero, sólo pueden el dueño del fichero y el root.
chown: Cambia el dueño de un fichero, hay que tener privilegios especiales para hacer esto.
chgrp: Cambia el grupo de un fichero, se puede cambiar a un grupo al que el dueño pertenezca.
chmod (otras opciones): chmod 0777 (otorga rwx a todos), -R (todos los ficheros y directorios del árbol).

} 4.1.3 GESTIÓN DE GRUPOS

Identificadores de grupo: Permiten que varios usuarios compartan una lista de permisos de manera práctica.
/etc/group: Fichero donde están definidos todos los grupos del sistema.
/etc/group (estructura): Nombre.contraseña:GID:nombres de usuarios miembros del grupo.
PAM (modelo de autenticación dinámica): Es un servicio que permite al administrador del sistema decidir cómo autentican a los usuarios/as de diferentes aplicaciones (/etc/pam.d).
Creación y eliminación de grupos: adduser username groupname, deluser username groupname, addgroup groupname, delgroup groupname.

[] TEMA 5: CONTROL DE VERSIONES (GIT)

[] 5.1 INTRODUCCIÓN A LOS SCV

} 5.1.1 REGISTRO DE CAMBIOS

Sistema de Control de Versiones (SCV): Programa que va registrando las modificaciones que hacemos sobre el proyecto y que va creando “fotos” del nuevo estado del proyecto tras cada modificación. Además, mantiene internamente un registro de todos los cambios efectuados en cada archivo (evolución y representación gráfica del proyecto) y permite volver a puntos anteriores para auditar los cambios, corregir los errores, etc.

} 5.1.2 CORRECCIÓN DE ERRORES Y PETICIÓN DE MEJORAS

Sistema de seguimiento de problemas (ITS): Registra avisos y conversaciones asociadas al procesamiento, enlazando los cambios registrados a la resolución de cada error o los que responden a una petición de mejora específica.

} 5.1.3 CLONADO, FORK Y CONTRIBUCIONES

Clonación de un proyecto: Se obtiene una copia local del mismo y se trabaja sobre ella.
Fork: Crea otra copia remota del proyecto que luego se pueda clonar localmente para mantener los cambios que se vayan realizando.
Pull request: Ocurre cuando un programador que está trabajando con sus copias puede solicitar la integración de sus cambios en el proyecto principal.

} 5.1.4 VERSIONES DE LANZAMIENTO (RELEASES)

Versiones (estructura): 4.2.1, donde 4 es el Major Number (cambios muy significativos que pueden romper cosas o no ser compatibles con versiones anteriores, si hay un 0 indica software inmaduro o inestable), 2 es el Minor Number (indica nuevas funcionalidades sin riesgo) y 1 es el Patch (indica resolución de problemas).

} 5.1.5 SCV CENTRALIZADO Y DISTRIBUIDO

SCV centralizado: Todos los participantes trabajan sobre la misma copia, sin embargo, no funciona cuando dos o más personas trabajan sobre el mismo archivo haciendo cambios diferentes e intentando actualizar la copia maestra con sus respectivas modificaciones.
SCV distribuido: Permite que cada persona trabaje con su copia local del proyecto completo y haga cambios en ella.

} 5.1.6 EJEMPLOS DE SCV

Git (2005): Velocidad, diseño sencillo, buen soporte para cambios en paralelo (desarrollo no lineal), totalmente distribuido, manejo de proyectos de gran tamaño (velocidad y transferencia de datos).

Mercurial (2005): Escalabilidad y alto rendimiento, completamente descentralizado, soporte para archivos de texto plano y binarios, líneas de desarrollo paralelas y capacidad avanzada de integración de cambios.
GitLab y GitHub: Servicio web para alojamiento y desarrollo de proyectos software (forja) con SCV, herramientas para DevOps, versión libre CE y versión privativa EE.

[] 5.2 SCV (GIT)

} 5.2.1 SCV
SCV (características): Sincroniza trabajo en archivos del proyecto; hace, deshace, mezcla (merge) y agrupa cambios (commit); tiene variantes y diferentes líneas de desarrollo (branches); preserva la historia del proyecto (grafo de cambios); evita el nombrado manual; desarrollo en grupo.

} 5.2.2 SCV DISTRIBUIDO
SCV distribuido: Varios árboles de ficheros; trabajo local y sincronización (conjunto de bosques de árboles de ficheros en diferentes máquinas).

} 5.2.3 ¿QUÉ NO ES GIT?
¿Qué no es Git?: No es un sistema general para hacer copias de seguridad, puede servir de backup para el fuente pero requiere intervención de usuario, y tampoco es bueno gestionando archivos binarios.

} 5.2.4 ¿QUÉ COSAS PUEDO HACER?
Funciones de Git: Añadir y modificar uno o varios ficheros; crear y mezclar una branch con y sin conflictos; ver el historial de cambios; deshacer un conjunto de cambios confirmados; compartir y sincronizar código con un repositorio remoto o central.

} 5.2.5 TERMINOLOGÍA BÁSICA
clone: Copia un repositorio diferente (remoto) a un sitio local.
commit: Conjunto de cambios que se aplican agrupados al repositorio local.
fetch/pull: Trae cambios de un repositorio diferente (o remoto), fetch trae y pull además hace merge.
push: Manda cambios a un repositorio diferente que puede ser remoto.
head: Referencia al commit sobre el que estamos trabajando.
branch: Etiqueta de una rama de código (línea separada de cambios).
master: Branch principal del repositorio (de trabajo o integración), en algunos sitios llamada main.

} 5.2.6 FUNCIONAMIENTO INTERNO
Hash (SHA): Función matemática que general un código hexadecimal que resume el contenido.
Elementos internos: tree (directorio interno con una lista de SHAs con los ficheros y directorios que contiene), fichero (blob de bytes), HEAD, etiquetas de branch, metadatos en el directorio .git.

} 5.2.7 CONFIGURACIÓN
git config user.name “name”: Configuración a nivel de proyecto (.git/config).
git config –global user.name “name”: Configuración a nivel global (proyectos del usuario, $HOME/.gitconfig),
git config –system user.name “name”: Configuración a nivel system (proyectos de la máquina, /etc/gitconfig).
Alias de comandos (ejemplo): git config –global alias.adog “log –all –decorate –oneline –graph”.
git config –global core.editor vim: Configuración del editor de texto (en este caso, vim).
git config –global merge.tool emerge: Configuración de herramienta para mezclar cambios (merge).
git config –global init.defaultBranch <nombre>: Define el nombre de la rama principal (master o main).

git rm fichero: Borra un fichero del repositorio, haciendo que no esté en el siguiente commit.

} 5.2.8 FICHERO .GITIGNORE

Fichero .gitignore: Raíz del proyecto, # para comentarios, *.a (ignorar ficheros acabados en .a), !lib.a (no ignora lib.a), /TODO (fichero /TODO en el raíz), build/ (ignorar todo lo de build), ** (ignora directorios y subdirectorios).

} 5.2.9 GIT DIFF

Tipos de git diff: git diff (entre work y stage), git diff –staged (entre stage y repo), git diff sha1 sha2 (entre dos objetos).

} 5.2.10 TIPOS DE REPOSITORIOS

Repositorio bare: No tiene directorio de trabajo ni stage, le vale a los servidores (push y pull de cambios), el directorio acaba en .git (git init –bare origen.git).
Repositorio clone: Clona un repositorio remoto, deja remote apuntando a ese repositorio (git clone <url>).

Ejemplo de simulación de servidor con dos clientes para probar conflictos y demás en local:

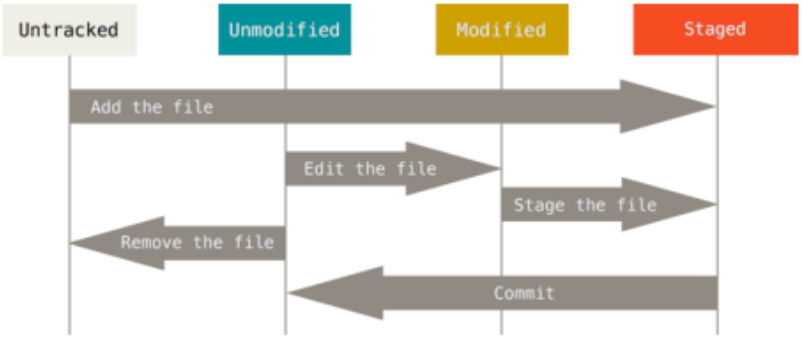
```
repocentral.git          trabajocasa          trabajolabo
$ cd /trabajo
$ git init --bare repocentral.git
$ cd /trabajo
$ git clone /trabajo/repocentral.git trabajocasa
$ cd /trabajo/trabajocasa
$ git remote -v
$ git clone /trabajo/repocentral trabajolabo
$ cd /trabajolabo
$ vi tub.c
$ git add tub.c
$ git commit -m 'primer fichero'
$ git push
$ cd /trabajo/trabajolabo
$ git pull
```

Ejemplo de simulación de servidor con dos clientes para probar conflictos y demás en remoto:

```
repocentral.git          trabajocasa          trabajolabo
$ ssh paurea@alpha.aulas.gsyc.urjc.es
paurea@alpha$ git init --bare repocentral.git
$ git clone paurea@alpha.aulas.gsyc.urjc.es:repocentral.git trabajocasa
$ cd trabajocasa
$ ssh paurea@alpha.aulas.gsyc.urjc.es
paurea@alpha$ git clone repocentral.git trabajolabo
paurea@alpha$ cd trabajolabo
$ vi tub.c
$ git add tub.c
$ git commit -m 'primer fichero'
$ git push
paurea@alpha$ git pull
```

[] 5.3 GIT AVANZADO

} 5.3.1 CICLO DE CAMBIOS EN ARCHIVO DEL PROYECTO



} 5.3.2 PROPÓSITOS DE LAS RAMAS

Rama: Versión alternativa del proyecto creada a partir de una instantánea (commit) del estado actual del proyecto.

Construcción del grafo de cambios: Commit con puntero a la instantánea de los cambios añadidos al stage, el commit normal no tiene ancestros, el normal tiene uno y el merge tiene dos o más.

} 5.3.3 CREACIÓN DE RAMAS

Creación de ramas: La rama creada por defecto es la main o master; al crear una nueva rama surge otro puntero que apunta al mismo commit que la rama actual; el puntero HEAD indica la rama actual de trabajo.

} 5.3.4 CAMBIOS Y COMMITS EN RAMAS

git checkout testing: Cambiar a la rama testing.
git log: Muestra el historial de cambios de la rama actual.
git log <branch>: Muestra el historial de cambios de una rama concreta.
git log -all: Muestra el historial de cambios de todas las ramas.
git switch testing: Cambia a la rama testing.
git switch -c <branch>: Crea la nueva rama <branch>.
git switch -: Cambia a la rama en la que se estaba previamente.

} 5.3.5 GESTIÓN DEL ÁREA DE TRABAJO

git commit –amend: Descarta el commit inicial y se reemplaza por completo con la nueva versión de cambios como si el anterior nunca hubiese ocurrido (no se deben aplicar a cambios que no se han enviado con git push, ya que se pueden crear problemas).
git reset HEAD <file>: Descarta los últimos cambios que hemos añadido al área de staging.
git checkout – <file>: Descarta por completo los últimos cambios en un archivo antes de añadirlo al área de staging (operación irreversible).
git restore –staged <file>: Deshace cambios en el staging area.
git restore <file>: Descarta cambios en un archivo del directorio de trabajo.

} 5.3.6 PULL REQUEST

Ejemplo de envío de cambios a un repositorio del que no tenemos privilegios de edición:

```
$ git clone
$ cd project
$ git checkout -b featureA
$ git commit
$ git commit
$ git remote add myfork
$ git push -u myfork featureA
```

git request-pull: Genera información sobre la petición que se puede enviar a los gestores del proyecto mediante un correo electrónico de contacto.

} 5.3.7 ETIQUETAS

git tag: Muestra todas las etiquetas de un repositorio.

git tag -l <tag>: Muestra las etiquetas que coincidan con el patrón de búsqueda.
Etiqueta LightWeight (git tag v1.4-lw): Rama sin recorrido (puntero) que sirve para marcar hitos internos o de poca importancia.
Etiqueta Annotated (git tag -a v1.4 -m <tag>): Se usa para hitos relevantes como versiones publicadas (checksum, nombre y correo del etiquetador, fecha, mensaje de descripción, firma segura).
git log –pretty=oneline: Anota commits anteriores al HEAD.
git push origin v1.5: Envía etiquetas al repositorio remoto mediante una operación git push.
git push origin –tags: Transfiere al repo remoto todas las etiquetas locales, tanto LightWeight como Annotated.
git push <remote> -follow-tags: Envía solo las etiquetas del tipo Annotated.
git tag -d <tag>: Borra etiquetas del repositorio local.
Otras alternativas: git push <remote> :refs/tags/<tagname> o git push origin –delete <tag>.

} 5.3.8 DIFF Y PATCH

git diff: Encuentra el mínimo número de diferencias entre dos ficheros o directorios, obteniendo como resultado un fichero patch con las diferencias.
git patch: Permite aplicar esas diferencias.

Ejemplo de salida del comando git diff:

```
diff --git a/otro.txt b/otro.txt
index 866b380..bc52249 100644
--- a/otro.txt
+++ b/otro.txt
@@ -1,4 +1,6 @@
```

866b380 y bc52249: SHA de la versiones del fichero.
100644: Permisos de los ficheros (rwx).
-: Contenido de a no presente en b.
+: Contenido que falta en a presente en b.
Comienzo y longitud del texto en a (-) y b (+).

git apply / patch apply: Aplica el patch sobre la versión antigua para generar la nueva.

} 5.3.9 MERGE Y CONFLICTOS

git merge: Saca el parche entre el antepasado común y el origen y lo aplica en el destino.
Conflicto: Surge cuando no se pueden calcular las diferencias con diff ya que las dos versiones están editando en la misma línea.
gitk: Muestra una versión más gráfica del grafo de cambios en git log.

[] 5.4 LISTADO DE COMANDOS GIT

} 5.4.1 CONFIGURACIÓN DE GIT

git config –global user.name “name”: Establece el nombre adjunto a commits y tags.
git config –global user.email “email”: Establece el correo adjunto a commits y tags.
git config –global color.ui auto: Habilita algo de colorización en la salida de git.

} 5.4.2 TRABAJO DIARIO

git status: Muestra el estado del directorio de trabajo.
git add <file>: Agrega un archivo al staging area.
git diff <file>: Muestra los cambios de un archivo entre el directorio de trabajo y el staging area.
git diff –staged <file>: Muestra cualquier cambio entre el staging area y el directorio de trabajo.
git checkout – <file>: Descarta los cambios en el directorio de trabajo (operación irreversible).
git reset <file>: Revierte el repositorio a un estado de funcionamiento conocido anteriormente.
git commit: Crea una nueva confirmación a partir de los cambios agregados al staging area.
git rm <file>: Elimina el archivo del directorio de trabajo y del staging area.
git stash: Guarda los cambios actuales en el directorio de trabajo para un uso posterior.
git stash pop: Aplica el contenido del stash almacenado en el directorio de trabajo y en el staging area.
git stash drop: Elimina un stash específico de todos los stash anteriores.

} 5.4.3 COMENZANDO UN PROYECTO

git init <project>: Crea un nuevo repositorio local, si se le proporciona un nombre se crea un repositorio en su interior.
git clone <url>: Descarga un proyecto con todo el historial desde el repositorio remoto.

} 5.4.4 MODELADO DE RAMAS EN GIT

git branch -a: Enumera todas las ramas, tanto locales como remotas.
git branch <branch>: Crea una nueva rama haciendo referencia al HEAD actual.
git checkout -b <branch>: Cambia el directorio de trabajo a la rama especificada, creándose en el caso de que ésta no exista.
git merge <branch>: Une la rama branch con la rama actual.
git branch -d <branch>: Elimina la rama seleccionada forzando su eliminación en el caso de que esté unida con otra.

} 5.4.5 REVISIÓN DEL TRABAJO REALIZADO

git log <-n count>: Muestra el historial de commits de la rama actual, con -n limita el conteo hasta el commit n.
git log –oneline –graph –decorate: Descripción general con etiquetas de referencia y gráfico del historial.
git log ref..: Lista de commits presentes en la rama actual y no fusionadas en ref (nombre del tag o rama).
git log ..ref: Lista de commits presentes en la referencia y no fusionadas con la rama actual.
git reflog: Lista de operaciones (checkouts o commits) realizadas en el repositorio local.

} 5.4.6 ETIQUETANDO COMMITS CONOCIDOS

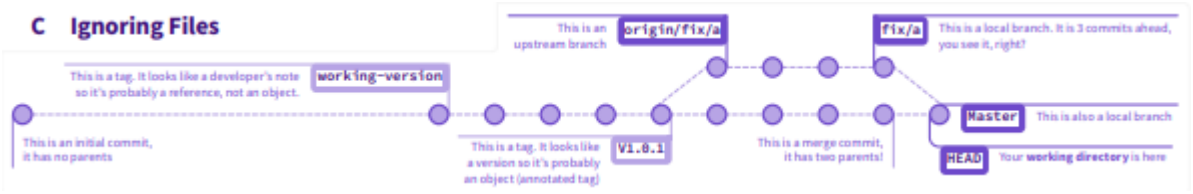
git tag: Lista todas las etiquetas.
git tag <tag> <SHA>: Crea una referencia de etiqueta llamada tag para el commit actual o uno específico.
git tag -a <tag> <SHA>: Crea un objeto de etiqueta llamado tag para la confirmación actual.
git tag -d <tag>: Elimina una etiqueta del repositorio local.

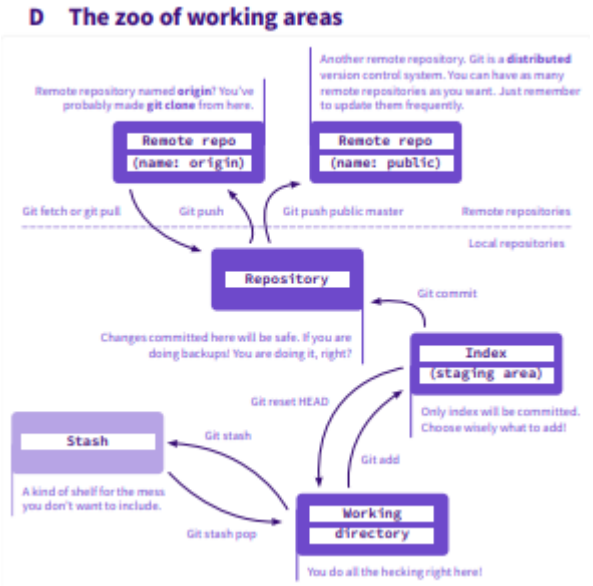
} 5.4.7 REVIRTIENDO CAMBIOS

git reset –hard <dest>: Cambia la rama actual a la referencia de destino dejándolo como uncommit, descartando todos los cambios.
git revert <SHA>: Crea un nuevo commit revirtiendo los cambios del commit especificado, generando una inversión de cambios.

} 5.4.8 SINCRONIZANDO REPOSITORIOS

git fetch <remote>: Obtiene cambios del repositorio remoto pero sin actualizar las ramas de seguimiento.
git fetch –prune <remote>: Elimina las referencias remotas eliminadas en el repositorio remoto.
git pull <remote>: obtiene cambios del repositorio remoto y fusiona la rama actual con su upstream.
git push –tags <remote>: Sube los cambios locales al repositorio remoto y envía las etiquetas.
git push -u <remote> <branch>: Sube la rama local al repo remoto estableciendo su copia como upstream.





[] TEMA 6: ARRANQUE DEL SISTEMA

[] 6.1 ARRANQUE DEL SISTEMA

} 6.1.1 SECUENCIA DE ARRANQUE EN UBUNTU Y LINUX

Proceso de arranque de un máquina Ubuntu: Firmware -> Cargador primario -> Cargadores secundarios -> Kernel -> Área de usuario (init).
Secuencia de arranque: Hardware (UEFI/BIOS y Bootloader) -> Espacio de Kernel (Kernel) -> Espacio de usuario (Init y otros espacios de usuario).

} 6.1.2 FIRMWARE

Firmware: Software que viene integrado en el hardware (memoria en la placa padre), hay distintos tipos.
Funciones principales: Inicializar el hardware de la máquina (enumerar dispositivos, inicializar controlador de memoria DRAM, gestión de energía, sistema de gestión del sistema); ofrecer una interfaz para interoperar con algunos componentes hardware; iniciar el arranque del sistema (de disco o red); normalmente se puede actualizar (proceso delicado); los PCs antiguos usan BIOS (Basic I/O System) y los más modernos usan UEFI (puede emular una BIOS en modo legacy).

} 6.1.3 UEFI

UEFI (Unified Extensible Firmware Interface): Conjunto de especificaciones desarrolladas por el UEFI forum que definen la arquitectura de la plataforma firmware para arranque del sistema y su interfaz con el sistema operativo. Tiene su propio intérprete de comandos y se pueden desarrollar aplicaciones efi (*.efi).
Preboot eXecution Environment (PXE): Entorno que permite cargar remotamente el sistema operativo a través de la red en lugar de leer el disco local.
Características: Tiene configurado un orden de arranque (CD/DVD y discos); todo disco con formato GPT debe tener una sola partición ESP que es en realidad una partición FAT; parted la llama boot y el instalador biosgrub; hay directorios con los cargadores de los sistemas operativos instalados en ese disco.
/boot/efi/EFI/BOOT/BOOTX64.EFI: Aplicación que se ejecuta por omisión (montada en /boot/efi).
efibootmgr -v: Muestra las variables de UEFI, la variable BootOrder indica el orden de ejecución de las aplicaciones EFI cada una con su dispositivo y ruta. Permite modificar variables, entradas, etc.

} 6.1.4 CARGADOR

Cargador: Distintos tipos (GRUB legacy y GRUB); gestores de arranque (permite seleccionar y lanzar otros cargadores para arrancar distintos sistemas operativos); objetivo (seleccionar y encontrar la imagen del kernel que se quiere arrancar en una partición del disco, cargarla en memoria y saltar a ella); distintas fases (cargador primario y secundario, …); cargador muy pequeño en BIOS y muy grande en UEFI; en UEFI no es estrictamente necesario usar un cargador; GRUB dispone de un intérprete de comandos propio y un driver para extender particiones EXT.

} 6.1.5 GRUB

GRUB: Configurado con la lista de imágenes del kernel que tenemos en las particiones de disco para que pregunte cuál debe arrancar o arranque una directamente sin mostrar menú; cargamos GRUB con Esc (UEFI) o con Shift (BIOS); se puede activar el intérprete de comandos de GRUB para realizar operaciones avanzadas como pasar parámetros al kernel; al final traerá la imagen del kernel a memoria, leerá y rellenará ciertas cabeceras hdr y saltará al punto de entrada del núcleo denominado linux setup.

} 6.1.6 LINUX SETUP

Linux setup: Es el código en Real Mode de Linux. Cuando un Intel/AMD arranca, éste está en modo de emulación del Intel 8086 16-bit, direcciones de 20 bits, menos de 1 MB de RAM disponible, sin paginación; a estas alturas puede que el resto del kernel esté comprimido; configura los segmentos de memoria, pila, montón, etc, y recolecta información sobre el hardware.

} 6.1.7 KERNEL

Kernel (instrucciones de arranque): Pasar la CPU a Protected Mode (32-bit con memoria virtual); inicializar consola, detectar memoria, inicializar teclado y vídeo; pasar la CPU a Long Mode (64-bit); descomprimir el resto del kernel y relocalizar el código, configurando tabla de interrupciones, etc; saltar a start kernel; configurar la CPU0; imprimir Linux Banner en la consola; reservar memoria para initrd (RAM disk de inicio); inicialización de entrada/salida, PCI, SMP, DMA, planificador, cachés y sistemas de ficheros VFS.
Creación de procesos: PID0 o cpu_idle (ejecuta cuando no hay nada que hacer); PID1 o init (más adelante); PID2 o kthreadd (gestiona la creación de hilos del kernel); después se activan el resto de CPUs.

} 6.1.8 INITRD

Initrd (Initial RAM Disk): Sistema de ficheros que se usa para poder arrancar el sistema completo, actualmente se usa initramfs que usa el driver tmpfs.
Se monta en el raíz de forma temporal porque hay muchos drivers distintos para todos los tipos de hardware de almacenamiento que existen (en algunos casos se requieren comandos). Son módulos del kernel que se necesitarán cargar para poder montar el sistema de ficheros raíz verdadero. Todo ese sistema de ficheros está en un fichero /boot/initrd.img-*, y con el comando unmkinitramfs se puede extraer uno a mano y ver cuál es su contenido. Una vez montado el initrd en /, el proceso con PID 1 sigue estos tres pasos:
1. Ejecuta el script /init para montar sysfs y procfs para crear distintas variables de entorno, ejecutar otros scripts de arranque, cargar módulos necesarios, gestionar volúmenes lógicos LVM, etc.
2. Ejecuta /sbin/run-init para montar el raíz de verdad y eliminar de la memoria el initrd.
3. Ejecuta /sbin/init (del raíz de verdad).

} 6.1.9 INIT

init: El proceso con PID 1 quedará ejecutando el programa /sbin/init hasta que se apague la máquina, en Linux es systemd y en otras es System V init (sysvinit), todos los procesos de usuarios son hijos de init.
Propósitos: Montar sistemas de ficheros configurados; crear procesos que terminan ejecutando la interfaz gráfica como programas de login; inicio y parada de servicios (demonios); libera recursos cuando mueren procesos; espera a que se ordene el apagado o reinicio del sistema (activación/desactivación de un servicio).

} 6.1.10 SYSTEMD

systemd: Colección de demonios, bibliotecas, programas y componentes del núcleo.
Funciones: Ejecución periódica de programas y de servicios en demanda; control de logins, bitácoras y conexiones de red; manejo de la configuración de los dispositivos que se conectan; información de localización (locales); maneja el concepto de unit.

Tipos de units: Service Units (controlan los servicios, demonios); Mount Units (controlan los sistemas de ficheros); Target Units (controlan otras units); Path Units (vigilan rutas para reaccionar si cambian ficheros o directorios).
pkg-config systemd –variable=systemdsystemunitdir o /lib/systemd/system: Indica la ruta donde se instalan las units. Son ficheros de texto plano que no se deben modificar (nombre-de-unidad.tipo).
pkg-config systemd –variable=systemduserunitdir o /etc/systemd/system: Indica la ubicación de las units de usuario, éstas sí se pueden modificar si es necesario y tienen más precedencia.
Tipos de dependencia de units: Requires (estricta, se activará y si falla la activación de la dependencia no se puede activar esta unit); Wants (se intentará activar la dependencia, pero si falla no importa); Requisite (debe estar activa ya, si no lo está falla la activación de esta unit); Conflicts (si está activa se desactiva ya que no es compatible con esta unit).
Modificadores de orden: Before (esta unit se activa antes de las units indicadas); After (se activará después de las indicadas).
Modificadores de condición: ConditionPathExists=p (si el fichero p existe se activa);
ConditionPathIsDirectory=p (si existe y es directorio).

} 6.1.11 COMANDOS DE SYSTEMD

systemctl: Permite controlar y dar órdenes a systemd.
systemd-analyze: Permite analizar y depurar systemd.
systemctl start nombre-de-unit: Así se arranca un servicio; si en lugar de start ponemos stop se para el servicio; si ponemos restart se está reiniciando el servicio; con reload podemos recargar la configuración.
systemctl enable nombre-de-unit: Así se habilita un servicio (se arranca automáticamente cuando se inicie el sistema); si en lugar de enable ponemos disable se deshabilita el servicio; se puede arrancar manualmente una unit que está deshabilitada.
systemctl status nombre-de-unit: Forma de comprobar el estado de un servicio; muestra un resumen del servicio (tiempo que lleva arrancado, fichero, información de procesos, últimas líneas de bitácora, etc).
systemctl list-units: Muestra todas las units activas (columnas con el nombre de la unidad, configuración cargada en memoria, estado y descripción); con –all muestras todas las units tanto activas como inactivas; con –state=estado muestra todas las units con ese estado.
systemctl list-unit-files: Muestra todos los ficheros de las units; static indica que la unit no puede ser deshabilitada porque es una unit que simplemente realiza una acción normalmente como dependencia de otra unit; masked indica que no se permite arrancarla.
systemctl show nombre-de-unit: Muestra las propiedades de la unit.
systemctl list-dependencies nombre-de-unit: Muestra las dependencias de la unit (Required o Wanted); con –all lo hace recursivamente; con –reverse muestra las units que dependen de esta unit.
systemctl edit -full nombre-de-unit: Sirve para editar el fichero de la unit; cuando se termina la edición se escribe el fichero de la unit en /etc/systemd/system que es el directorio con más precedencia para las units.
systemctl daemon-reload: Recarga systemd.
systemctl halt: Apaga el sistema; con poweroff hace un apagado total; con rescue pone el sistema en modo single-user para recuperación.
systemctl poweroff: runlevel 0 (apagar).
systemctl rescue: runlevel 1 (single-user).
systemctl isolate multi-user.target: runlevels 2,3 y 4 (multi-user).
systemctl isolate grafical.target: runlevel 5 (multi-user con entorno gráfico).
systemctl reboot: runlevel 6 (reinicio).

} 6.1.12 RUNLEVELS

Runlevel: Describe una configuración del sistema para un modo de operación; cuando se activa un runlevel se paran todos los servicios que no hacen falta y se activan los que sí hacen falta; systemd lo hace con sus targets.

} 6.1.13 ENTORNO GRÁFICO

Entorno gráfico: Es lo último que se arranca en un sistema de escritorio. Suele arrancarse un programa que es la pantalla gráfica de login (gdm, sddm o lightdm) y éste arranca el sistema de ventanas xorg que a su vez

ejecuta una sesión con el manejador de ventanas y menús (unity-session, gnome-session o lxsession). Los xorg pueden arrancarse a mano con startx desde una consola de texto.
Direct Rendering Manager (DRM): Subsistema del kernel que implementa la interfaz con la GPU.
Wayland: Gestiona directamente los permisos de compartición de recursos gráficos (compartir pantalla); diseño más ligero y mantenible que en xorg.

[] TEMA 7: FICHEROS Y COMANDOS AVANZADOS

[] 7.1 FICHEROS Y COMANDOS AVANZADOS

} 7.1.1 METADATOS E I-NODOS
Metadatos: Son los datos sobre un fichero y no los contenidos en él (nombre y permisos de un fichero).
Directorio Unix: Lista de entradas, las cuales contienen el nombre del fichero y un número que lo identifica (i-nodo). Un mismo i-nodo puede tener distintas entradas (. para el i-nodo y .. para el i-nodo padre) para distintos nombres, es decir, pueden tener múltiples referencias al mismo fichero, y se pueden consultar con el comando ls -i.
I-nodo: Estructura que está en la partición del sistema de ficheros, cuyo número se utiliza para localizar su estructura (el directorio raíz tiene número de i-nodo 2), la cual contiene los metadatos del fichero y la lista de bloques de disco que contienen los datos del fichero.
I-nodo (estructura): Permisos, tiempos (acceso a datos con itime, modificación de los datos con mtime, modificación del i-nodo con ctime), tamaño, dueño, tipo, número de bloques, contador de referencias (links), no contiene el nombre del fichero.

} 7.1.2 ENLACES DUROS
Enlace duro: Es otro nombre para el fichero o entrada de directorio para el i-nodo.
Contador de referencias: Incrementa cuando se crea un nombre nuevo para el fichero, decrementa cuando se elimina una entrada de directorio que hace referencia al i-nodo, cuando el contador llega a 0 se puede eliminar el fichero.
Desventaja: No se pueden crear enlaces duros para directorios, ya que rompen la jerarquía, y crean ambigüedad con el padre (..).
ln: Permite crear un enlace duro sin la opción -s.

} 7.1.3 TIPOS DE FICHEROS
Tipos de ficheros: Enlaces simbólicos (symlink o ‘l’), pipes con nombre (fifo o ‘p’), dispositivos (device o ‘c’ o ‘b’), conexiones de red (sockets o ‘s’).

} 7.1.4 ENLACES BLANDOS
Enlace blando: Fichero especial distinto (symlink) cuyos datos contienen la ruta al fichero enlazado, pueden romperse si el fichero enlazado se borra, y pueden crearse con el comando ln -s.

} 7.1.5 PIPES CON NOMBRE
Pipes con nombre (fifo): Pipe con nombre que persiste en el tiempo, se crean con el comando mkfifo y se borran con el comando rm.

} 7.1.6 DISPOSITIVOS
Dispositivos: Se presentan como ficheros en Unix, pueden usarse las mismas herramientas que usamos con los ficheros convencionales, se encuentran en /dev siguiendo un esquema de nombrado, no es común tener que crearlos ya que suelen crearse automáticamente (el demonio udev se encarga de esta gestión) y siguen la estructura /dev/sd<posiciónBus>, donde posiciónBus puede ser a, b, etc.
Dispositivos de caracteres: Trabaja con flujos de bytes.
Dispositivos de bloques: Trabaja con trozos de datos de cierta longitud (bloque), tienen un tamaño determinado y se accede aleatoriamente a sus bloques (discos duros).
Números de identificación: Major number (indica la clase del dispositivo, todos los de la misma clase tienen el mismo), minor number (indica la instancia concreta dentro de esa clase de dispositivo).

mknod: Crea dispositivos, proporcionándoles el tipo (b,c,etc) y los dos números (major y minor number).

} 7.1.7 EL COMANDO DD

dd: Útil para copiar datos entre dispositivos de bloques.
dd (modificadores): if=file (fichero de entrada), of=file (fichero de salida), bs=size (tamaño del bloque), count=num (número de bloques a transferir), skip=num (número de bloques a saltar de la entrada), seek=num (número de bloques a saltar en la salida).

} 7.1.8 EL COMANDO RSYNC

rsync: Permite copiar ficheros locales o remotos de manera rápida y versátil, donde en vez de copiarlos ciegamente compara los directorios fuente y destino y sólo se copian los elementos que han cambiado.
rsync local: rsync [option] [src] [dest]
rsync remoto: Pull (rsync [option] [user@]host:src [dest]), push (rsync [option] src [user@]host:dest).

} 7.1.9 INSPECCIÓN DEL SISTEMA

Formas de inspeccionar el sistema: Navegación por sistemas de ficheros sintéticos (/proc) o ejecutando comandos especializados (ps).

} 7.1.10 COMANDOS DE INSPECCIÓN DEL SISTEMA

/proc: Sistema con ficheros generados al vuelo, tiene un directorio por cada proceso de ejecución.
/proc (elementos): cmdline (línea de comandos que ejecuta), cwd (enlace simbólico a su directorio actual), environ (variables de entorno), exe (enlace simbólico al ejecutable), fd (ficheros que tiene abiertos), io (información sobre entrada/salida), maps (regiones de memoria), mem (memoria del proceso).
/proc (ficheros y directorios de interés): cpuinfo (información sobre la CPU), kmsg (log del kernel), meminfo (información sobre el uso de la memoria), modules (módulos cargados), net (directorio con información sobre la red), uptime (tiempo que lleva levantado el sistema).
/sys: Interfaz para acceder a las estructuras internas del kernel (directorio con ficheros sintéticos).
/sys (opciones): block (enlaces simbólicos para cada dispositivo de bloques), class (directorios para cada clase de dispositivo), devices (representación del árbol de dispositivos), firmware (interfaz para manipular objetos y atributos del firmware), fs (interfaz para controlar los sistemas de ficheros), kernel (manipulación variada del kernel), module (módulos cargados), power (manipulación de la gestión de energía).
ps: Lista los procesos del sistema, comúnmente se ejecuta con los modificadores aux.
top: Muestra los procesos refrescando sus datos (consumo de CPU y memoria de los procesos).
top (orden de filas): h o ? (opciones interactivas), q (sale), P (uso de CPU), M (consumo de memoria).
lsof: Muestra los ficheros que tienen abiertos los procesos.
lsof (estructura): Comando, PID, usuario, descriptor de fichero, tipo, dispositivo que lo sirven, tamaño, posición y nombre del fichero.
mount: Muestra los sistemas de ficheros que están montados, algunos se encuentran en /etc/fstab.
mount (estructura): Dispositivo, punto de montaje, tipo de sistema de ficheros y opciones de montaje.
df: Muestra el espacio libre de los distintos sistemas de ficheros montados, con -h da los tamaños en un formato más legible y en potencias de 2 (sistema de ficheros, tamaño, espacio usado, dispositivo, %uso, ubicación de montaje), con -i lista la información de los i-nodos en lugar del uso de bloques en sistemas de ficheros.
netstat: Ofrece información sobre las conexiones de red.
netstat (opciones): -a (todas las conexiones), -at (conexiones TCP), -au (conexiones UDP), -tnl (puertos de la máquina que están escuchando servicios), -nr (tabla de enrutamiento de la máquina).
dmesg: Muestra el kernel ring buffer (mensajes escritos por el kernel) que tienen varios niveles de importancia (información, avisos, errores de distinta gravedad, etc), con -w se queda esperando nuevos mensajes.

} 7.1.11 USUARIOS

Usuarios: w (muestra los usuarios y la terminal que están usando el sistema en el momento), who (igual que w pero menos completo), last (muestra los últimos usuarios que han entrado en el sistema tanto remota como localmente).

} 7.1.12 LOGS

Logs: Están en /var/log (hay binarios y de texto), cuando llegan a un cierto tamaño se comprimen y se renombran asignándoles un número (kern.log.2.gz), el log de systemd se puede inspeccionar con el comando journalctl. Estos son algunos de los logs más relevantes:
syslog: Información general del sistema sin temas de autenticación, recopila los mensajes de distintos servicios y programas y los escribe en este log (es configurable).
auth.log: Información sobre todas las autenticaciones, tanto correctas como fallidas.
kern.log: Mensajes del kernel.
wtmp: Contiene la información que muestra el comando last.
dpkg.log: Mensajes sobre la instalación y gestión de paquetes Debian.
Xorg.N.log: Mensajes de la interfaz gráfica en la sesión N.

} 7.1.13 TRABAJO EN REMOTO

ssh: permite trabajar con una shell en un sistema remoto de forma segura (túnel de comunicación cifrado).
scp: Permite copiar un fichero desde el servidor remoto a la máquina local y viceversa.
scp (remoto-local): scp tu-login@servidor:fichero-origen fichero-destino
scp (local-remoto): scp fichero-origen tu-login@servidor:fichero-destino

[] TEMA 8: COMPILACIÓN, CARGA Y DEPURACIÓN DE PROGRAMAS

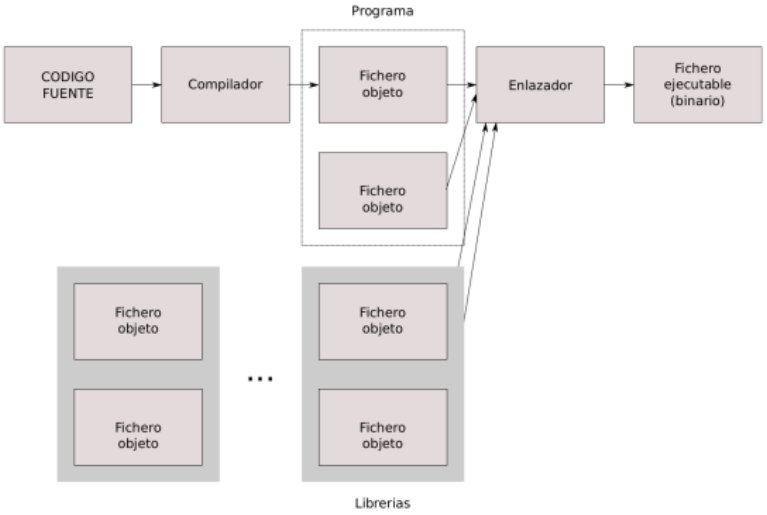
[] 8.1 COMPILACIÓN, CARGA Y DEPURACIÓN

} 8.1.1 ELF

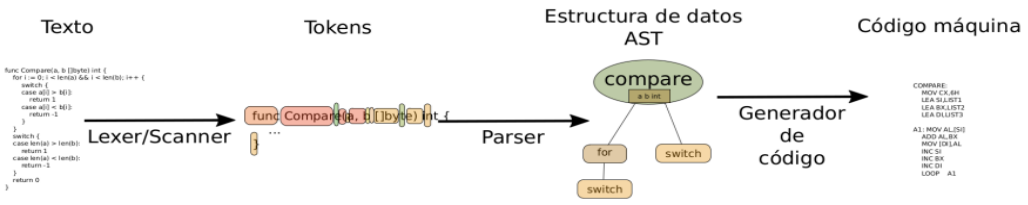
ELF (Executable and Linkable Format): Formato binario de fichero para ejecución nativa en Linux.

} 8.1.2 COMPILACIÓN

Librerías/bibliotecas: Funciones ya programadas, sus funciones y variables pueden ir dentro del binario final (enlazado estático) o en un fichero separado (enlazado dinámico).
Programas con varios ficheros: Pueden organizarse en módulos o paquetes, son parecidos a una librería pero sin empaquetar (herramienta Setuptools de Python).



Código relocalizable: Todos los saltos, direcciones de memoria, etc, son relativos (el programa se puede poner en varios sitios y funciona).
Ejemplo: Código relocalizable (JMP START+0x34, posición relativa a la dirección de comienzo del programa), código no relocalizable (JMP 0x344c1cd3).
Abstracción de proceso: Empieza por el programa de memoria que empieza en 0, todos los programas creen tener toda la memoria, salta a direcciones concretas cuando el programa está ejecutando.
Compilador: Traduce lenguaje de alto nivel a código máquina, construye un árbol de sintaxis AST y luego lo traduce, resuelve todos los símbolos definidos en el fichero creando una tabla, los símbolos resueltos depuran y dan mensajes de error y los no resueltos se resuelven más adelante con el enlazado, genera ficheros objeto, código relocalizable y es muy posible que tenga símbolos sin resolver.



} 8.1.3 ENLAZADO

Enlazado en tiempo de compilación (enlazado estático): El fichero ejecutable tiene todos los símbolos resueltos y es autosuficiente.
Enlazado en tiempo de ejecución (enlazado dinámico): El fichero ejecutable tiene símbolos que se resuelven al ejecutar dependiendo de librerías dinámicas.
Comandos: nm (imprime la tabla de símbolos), strip (quita la tabla de símbolos).
Carga: Copia a memoria y salta al punto de entrada (registro del PC) para ejecutar un binario, si quiere estar en un punto concreto de la memoria hay que copiarlo ahí y si tiene enlazado dinámico hay que cargar también las librerías dinámicas y resolver los símbolos (ld.so y ld-linux.so).
Carga en demanda: Copia del fichero binario siempre que se necesite.

} 8.1.4 LIBRERÍAS

Librerías dinámicas y objetos: Tienen un formato especificado por Elf, su tabla de símbolos está en Dwarf, las librerías dinámicas acaban en .so y las estáticas en .a.
Ubicación de las librerías dinámicas: Si no tienen un path absoluto busca en LD_LIBRARY_PATH, después en /etc/ld.so.cache, y si no en /lib o /usr/lib.
readelf -d fichero: Muestra las dependencias de un fichero.
ldconfig: Hace la configuración de la caché y los enlaces simbólicos de las librerías dinámicas (su configuración está en /etc/ld.so.conf y /etc/ld.so.conf.d).

} 8.1.5 DEPURACIÓN

Tipos de errores: Compilación, ejecución, funcionales, problemas de rendimiento y liberación de recursos.
Estrategias de depuración: Volcar el estado del programa antes y después de que pase a ser incorrecto; ir dividiendo en 2 el código para una buena convergencia con el número de líneas, instrumentar las estructuras de datos que impriman su estado, tener niveles de depuración habilitables como verbose, usar herramientas profesionales de depuración y un logger (librería que crea archivos de bitácora de manera profesional).
Depurador: Programa que permite parar, inspeccionar (core) o desensamblar el programa.
Depurador (elementos): Breakpoint (para cuando llega a un punto), watchpoint (para cuando se accede a una variable).

} 8.1.6 PROFILING

Profiling: Se usa la herramienta profiler, que mide el uso de recursos, memoria y búsqueda de puntos calientes, se pueden hacer a mano midiendo tiempo contando pasos por un sitio o asignación y liberación de recursos.
Tipos de profiling: Traza estadística (mide dónde está cada cierto tiempo), inyector de contadores de tiempo en las funciones, de asignación/liberación de recursos en memoria o hardware.

} 8.1.7 ANÁLISIS ESTÁTICO Y DINÁMICO

Análisis estático (linter): Encuentra errores y problemas automáticamente, hace análisis estático del código, puede poseer patrones incorrectos o fácilmente erróneos de uso y peligro de falsos positivos.
Análisis dinámico (Valgrind): Ayuda a encontrar problemas y errores automáticamente, hace análisis dinámico del código en ejecución instrumentado o sin instrumentar y pueden tener leaks (corrupción de memoria y patrones incorrectos del uso de recursos).

[] TEMA 9: CONTENEDORES VIRTUALES

[] 9.1 VIRTUALIZACIÓN (CONTENEDORES)

} 9.1.1 CONTENEDORES

Contenedor: Espacios de usuario donde ejecutan los programas que funcionan sobre el mismo kernel (sin necesidad de utilizar un hypervisor), además de ser un paquete que contiene la aplicación o aplicaciones a ejecutar más todas las dependencias necesarias, por lo que se ejecuta directamente en el kernel de forma aislada.
Es una tecnología con ventajas similares a los VMs pero con mejor aprovechamiento de los recursos (tardan milisegundos en arrancar y solo consumen la memoria que necesita la aplicación o aplicaciones en ejecución).

} 9.1.2 TIPOS DE DOCKER SEGÚN SU UBICACIÓN

Docker Hub: Repositorio público de imágenes Docker en el que se pueden publicar contenedores pre-configurados.
Docker en máquinas virtuales: Más pesadas, varios procesos, conexión por ssh, más seguridad al estar aislada del host.
Docker en contenedores: Más ligeras, un único proceso, acceso directo al contenedor, menos seguridad al ejecutarse como procesos en el host.

} 9.1.3 EJECUCIÓN DE CONTENEDORES SEGÚN SU UBICACIÓN

Ejecución de contenedores en Linux: Tecnología muy madura y disponible en cualquier distribución Linux.
Ejecución de contenedores en Windows: Tecnología mucho más reciente y disponible en sus últimas versiones.
Desarrollo con contenedores: En Linux se ejecutan de forma nativa, mientras que en Windows y MacOS se crean virtualmente (Docker Toolbox y Docker for Windows or Mac).

} 9.1.4 CLASIFICACIÓN DE DOCKERS

Docker Engine: Permite gestionar imágenes y contenedores, se accede desde la línea de comandos o mediante API REST, y puede gestionar Docker localmente, en VM o en plataforma cloud.
Docker Image: Plantilla básica para un contenedor (SO, bibliotecas, frameworks básicos y aplicaciones), se puede utilizar para ejecutar un contenedor, en Docker Hub hay muchas imágenes disponibles.
Docker Container: Se crea partiendo de una imagen equivalente a una VM. Cuando actualizamos algo dentro del contenedor no se modifica la imagen original, solo el contenedor.

} 9.1.5 OTROS TIPOS DE CONTENEDORES

Contenedores para servicios: Puede tener sentido pararlos y reanudar después la ejecución, ya que se usan constantemente (accesibles para un puerto en una IP local o localhost).
Contenedores para comandos: Ejecutan un comando y guardan los resultados en el host y se borran al finalizar la ejecución (obtienen ficheros procesados por el host y el resultado se queda en el propio host).
Docker Registry: Es un repositorio de imágenes Docker, que pueden ser públicas o privadas.