EXPLICACION DE PRACTICA I

¿Qué es un SO?

Es la parte esencial de cualquier sistema de cómputo.
Es un programa que actúa, en principio como intermediario entre el usuario y el hardware.
Su propósito: crear un entorno cómodo y eficiente para la ejecución de programas.
Su obligación: garantizar el correcto funcionamiento del sistema
Funciones principales:

- Administrar la memoria.
- Administrar la CPU.
- Administrar los dispositivos.

GNU/Linux: SO tipo Unix, distribución.
En 1983 Richard Stallman inicia el sistema GNU. Para asegurar que el mismo fuera libre, se necesitó crear un marco regulatorio GPL (General Public Licence).
En 1985 Stallamn crea la FSF (Free Software Foundation) con el fin de financiar al proyecto GNU.
En 1989 se crea la GPL.
En 1992 Linux Torvalds y Stallman deciden fusionar el kernel Linux (el cual ya se distribuía bajo la licencia GPL) y GNU, es allí donde nace GNU/Linux.

GNU = **G**NU **N**o es **U**nix
Se refiere a 4 libertades principales de los usuarios de software:

- Libertad de usar el programa con cualquier propósito.
- Libertad de estudiar su funcionamiento.
- Libertad para distribuir sus copias.
- Libertad para mejorar los programas.

Software libre:

- “Generalmente” de costo nulo.
- Se puede usar y redistribuir libremente.
- Corrección más rápida ante fallas.
- “Generalmente” se distribuye junto con su código fuente.
- Características que se refieren a la libertad de los usuarios para ejecutar, copiar, distribuir, estudiar, cambiar y mejorar el software.

Software propietario:

- Generalmente tiene un costo asociado
- No se lo puede distribuir libremente
- Generalmente no permite su modificación
- Normalmente no se distribuye junto con su código fuente
- Corrección de fallas está a cargo del propietario
- Generalmente al contar con una inversión mayor, proveen más avances y funcionalidad

- Menor necesidad de técnicos especializados

GPL: General Public License
Creada en 1989 por la FSF, su objetivo principal es proteger la libre distribución, modificación y uso del software GNU.
Su propósito es declarar que todo software publicado bajo esta licencia es libre y protegerlo teniendo en cuenta las 4 libertades principales.
Versión actual de la licencia: 3.

GNU/Linux – Características generales:
- Multiusuario.
- Multitarea y multiprocesador.
- Altamente portable.
- Posee diversos interpretes de comando y algunos de ellos son programables.
- Permite el manejo de usuarios y permisos.
- Todos es un archivo (hasta los dispositivos y directorios).
- Cada directorio puede estar en una partición diferente (ejemplos /temp /home, etc..).
- Es case sensitive.
- Es de código abierto.

Diseño
Fue desarrollado buscando la portabilidad de los fuentes.
Desarrollo en capas donde cada capa actúa como una caja negra hacia las otras, posibilitando el desarrollo distribuido.
Soporta diversos FileSystems.
Memoria virtual = RAM + SWAP
Desarrollado mayoritariamente de C y Assembler, también otros como java, perl, python, etc…

Estructura básica del SO
- **Kernel (núcleo)**
  Ejecuta programas y gestiona dispositivos de hardware.
  Es el encargado de que el software y el hardware puedan trabajar juntos.
  Sus funciones más importantes son la administración de memoria y CPU.
  En un núcleo monolítico hibrido:
  - Los drivers y código del kernel se ejecutan en modo privilegiado.
  - Lo que lo hace hibrido es la posibilidad de cargar y descargar funcionalidad a través de módulos.
- **Interprete de comandos**
  También conocido como CLI (Command Line Interface).
  Cada usuario tiene una interfaz o shell (se puede personalizar, son programables).
- **Sistema de archivos**
  Organiza la forma en que se almacenan los archivos en dispositivos de almacenamiento (fat, ntfs, ext2,…).
  Directorios importantes:
  - **/** tope de la estructura de directorios.
  - **/home** archivos de usuario.

- **/var** información que varía de tamaño (logs, BD, spools).
- **/etc** archivos de configuración.
- **/bin** archivos binarios y ejecutables.
- **/dev** enlace a dispositivos.
- **/usr** aplicaciones de usuarios.

- **Utilidades**
  Paquete de software (varia en las diferentes distribuciones). Ej: editores de texto, herramientas de networking, paquetes de oficina, interfaces graficas.

EXPLICACION DE PRACTICA II

MBR - (Master Boot Record)
Registro Maestro de Arranque. Primer registro del disco duro, el cual contiene un programa ejecutable y una tabla donde están definidas las particiones del disco duro.
Es sector reservado del físico (Cilindro 0, Cabeza 0, Sector 1). Existe un MBR en todos los discos.
Si existiese más de un disco rígido en la maquina, solo uno es designado como **Primary Master Disk**.

- Tamaño del MBR = tamaño estándar del sector del disco = 512bytes.
  - Los primero bytes 446 bytes corresponden al MBC (Master boot code).
  - 64 bytes son asignados para la tabla de particiones.
  - Al final existen 2 bytes libres.

MBC – (Master Boot Code)
El MBC es un pequeño código que permite arrancar el SO.
La última acción del BIOS es leer el MBC, lo lleva a memoria y lo ejecuta.

Proceso de arranque (Bootstrap)
En las arquitecturas x86, el BIOS es el responsable de iniciar la carga del SO a través de MBC
Carga el programa de booteo (desde el MBR)
El gestor de arranque lanzado desde el MBC carga el kernel

- Prueba e inicializa los dispositivos
- Luego pasa el control al proceso init

Gestor de arranque – (Bootloader)
Su finalidad es cargar una imagen de Kernel de alguna partición para su ejecución.
Se ejecuta luego del código de la BIOS.

GRUB (Gran Unifed Bootloader) – (Gestor de arranque)
Es el gestor de arranque múltiple mas utilizado.

- En el MBR solo se encuentra la **fase 1** del GRUB que solo se encarga de cargar la fase 1.5.
- La **fase 1.5** se encuentra en los 30kb siguientes del disco y carga la fase 2.
- La **fase 2** almacenada en disco, presenta una interfaz al usuario y carga el kernel seleccionado.

Se configura a través del archivo **/boot/grub/menu.lst**
GRUB 2
Actualmente el desarrollo está enfocado a grub2, y se está comenzando a utilizar en la mayoría de las distribuciones.

En GRUB2 la fase 1.5 ya no existe más.
Se configura a través del archivo **/boot/grub/grub.cfg** y no debería editarse manualmente (usar comando update-grub).

Particiones
Cada partición se formatea con un tipo de File System distinto (fat, ntfs, ext, etcétera…).
Debido al tamaño acotado del MBR para la tabla de particiones:
Se restringe a 4 la cantidad de particiones primarias.
O 3 particiones primarias y una extendida.

- Partición primaria: División cruda del disco, se almacena información de la misma en el MBR.
- Partición extendida: Sirve para contener unidades lógicas en su interior. Solo puede existir una partición de este tipo por disco. No se define un tipo de File System directamente sobre ella.
- Partición lógica: Ocupa la totalidad o parte de la partición extendida y se le define un tipo de File System directamente.

Emuladores / Virtualizadores
Permite que en un equipo puedan correr varios SO en forma simultánea compartiendo recursos de hardware.
Básicamente se pueden considerar 3 tipos (ordenados por eficiencia creciente):

- Emulación
    - Emulan hardware.
    - Tienen que implementar todas las instrucciones de la CPU.
    - Es muy costosa y poco eficiente.
    - Permite ejecutar arquitecturas diferentes a las soportadas por el hardware.
- Visualización completa
    - Permite ejecutar SO huéspedes en un sistema anfitrión (host).
    - Utilizan en el medio un hypervisor o monito de maquinas virtuales.
    - El SO huésped debe estar soportado en la arquitectura anfitriona.
- Paravirtualizacion
    - Permite correr SOs modificador exclusivamente para actuar en entornos virtualizados.

<u>Proceso de arranque System V</u>
Bootstrap

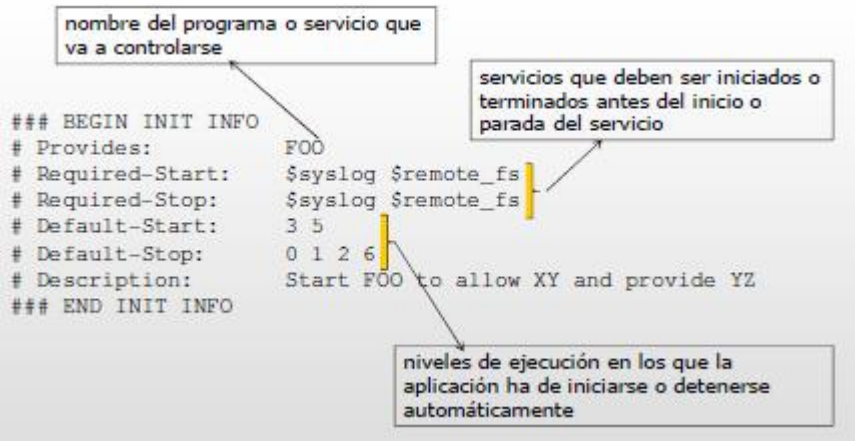
- Paso 0: Se ejecuta el código de la BIOS
- Paso 1: El hardware lee el sector de arranque (MBR)
- Paso 2: Se carga el gestor de arranque (MBC)
- Paso 3: Se carga el kernel
- Paso 4: Se monta el sistema de archivos raíz
- Paso 5: Se ejecuta el proceso init
- Paso 6: Lee el /etc/inittab
- Paso 7: Ejecuta los scripts apuntados por el runlevel 1
- Paso 8: El final del runlevel 1 le indica que vaya al runlevel por defecto
- Paso 9: Ejecuta los scripts apuntados por el runlevel por defecto
- Paso 10: El sistema está listo para usarse

Proceso init
Su función es cargar todos los subprocesos necesarios el correcto funcionamiento del SO.
Posee PID 1 y se encuentrar en **/sbin/init**
Se lo configura a través del archivo **/etc/inittab**
Es el encargado de montar los File Systems y de hacer disponibles los demás dispositivos.
Runlevels
El proceso de arranque lo dividimos en niveles. Cada uno es responsable de levantar o bajar una serie de servicios. Se encuentran definidos en **/etc/inittab** (id : niveles_de_ejecucion : acción : proceso): Existen 7, y permiten iniciar un conjunto de procesos al arranque, según el estándar:

0. Halt (Parada)
1. Single user modo (modo mono-usuario)
2. Multiuser, without NFS (modo multiusuario sin soporte de red)
3. Full multiuser (modo multiusuario completo, consola)
4. No se utiliza
5. X11 (modo multiusuario completo, con login grafico basado en X)
6. Reboot (Reiniciar)

Los scripts que se ejecutan están en **/etc/init.d/**
En **/etc/rcX.d** (donde X=0..6) hay links a los archivos de **/etc/init.d**
Formato de los links: **[S|K]<orden><nombreDelScript>**
S: Lanza el script con el argumento de start
K: Lanza el script con el argumento de stop

INSSERV
Se utiliza para manejar y actualizar el orden de los enlaces simbólicos del /etc/rcX.d en forma dinámica.



<u>Proceso de arranque Upstart</u>
Las tareas y servicios son ejecutados ante eventos (arranque del equipo o inserción de un dispositivo USB).
Las tareas se almacenan en el directorio **/etc/init**
Son scripts en texto plano que define las acciones a ejecutar.
El init es el responsable de lanzar las tareas.

Fstab
Define que particiones se montan al arranque. Su configuración se encuentra en **/etc/fstab**

Opciones:

- User: Cualquier usuario puede montar la partición.
- Noauto: No será montada al arranque del sistema.
- Ro: Read only, rw (read y write)

EXPLICACION DE PRACTICA III

Shell script

Es un archivo con una secuencia de comandos de determinado Shell (usaremos el **bash**).
Un script **bash** contiene:

- Comandos.
- Manejo de variables (no es fuertemente tipado).
- Comentarios #
- Estructuras de control
- Declaración de funciones.
- Puede usar código definido en otros scripts (comando **source**).

No requiere la instalación de ningún intérprete o compilador (todo GNU/Linux tiene al menos un shell).

Comandos útiles

cat *archivo*	Imprime el contenido de un archivo
echo *"hola mundo"*	Imprime texto
read var	Leer una línea desde entrada estándar y guardarla en la variable var.
cut *–d: -f1*	Quedarme con la primera columna de un texto. Parámetros: -d separador -f num_fila
wc *-l*	Contar la cantidad de líneas que se leen de la entrada estándar. Parámetros: -l indica contar líneas
grep *pepe /temp/**	Buscar todos los archivos que contengan la cadena pepe en el directorio /temp

Empaquetadores y compresores

Nomenclaturas:

- *.tar paquete de archivos (empaquetamiento)
- *.gz archivo comprimido

- Empaquetador: **tar** [opciones] [archivos]

El comando **tar** es utilizado normalmente para empaquetar o desempaquetar ficheros, empaquetar significa guardar en un único fichero una lista de varios ficheros, o el contenido de todo un directorio (o varios directorios).
Opciones:

- C: crea un nuevo archivo tar.

- V: Modo verbose, mostrara por pantalla las operaciones que va realizando por archivo.
- X: Extrae los archivos del tar.
- T: Muestra el contenido del archivo tar.
- F:
    - Cuando se usa –c: usa el nombre del archivo especificado para la creación del archivo tar.
    - Cuando se usa –x: retira del archivo el archivo especificado.
- Z: Comprime el archivo tar con gzip

Ejemplos:
Empaquetar en "archivo.tar": tar –cvf archivo.tar archivo1 archivo2 archivo3
Desempaquetar: tar –xvf archivo.tar
Empaquetar y comprimir: tar –cvzf archivo.tar archivo1 archivo2

- Compresor: **gzip** [opciones] [archivos]

Hay una herramienta especializada en la compresión y descompresión, es la herramienta gzip. Normalmente, el fichero a comprimir se reemplaza por otro con la extensión.gz, manteniéndose los mismos permisos, propietarios y tiempos de modificación. La cantidad de compresión obtenida depende de varios factores, típicamente, texto o código fuente se reduce en un porcentaje del 60 al 70%.
Ejemplos:
Compresión sobre un único archivo (resultante archivo.tar.gz): **gzip archivo.tar**
Descomprimir: **gzip –d archivo.tar.gz**

Redirecciones

- Redirección destructiva mediante el "**>**", si el archivo no existe se lo crea, si existe se pisa.
- Redirección NO destructiva mediante el "**>>**", funciona como un append.

Entrada / Salida / Error estándar

Todo programa en GNU/Linux tiene 3 archivos abiertos con un número de descriptor:
- 0 entrada estándar (por defecto teclado).
- 1 salida estándar (por defecto pantalla).
- 2 error estándar (por defecto pantalla).

Es posible volcar los errores a un archivo con **2>**:

```
find . –name foo 2> errores.txt
```

Uso de pipes

El "|" nos permite comunicar dos procesos por medio de un pipe o tubería desde la shell. El pipe conecta **stdout** (salida estandar) del primer comando con la **stdin** (entrada estándar) del segundo comando.

```
cat /etc/passwd | cut -d: -f1 | grep a | wc -l
```

Shell script

Para ejecutar un script tenemos varias formas de hacerlo:

- Desde línea de comandos podemos ejecutarlo pasándoselo al comando bash como argumento: **bash miscript.sh**
- Le podemos dar permisos de ejecución e invocarlo:
    - utilizando su path relativo: **./miscript.sh**
    - o mediante su path absoluto: **/home/yo/miscript.sh**

- podemos también ejecutarlo en modo debug (depuración): **bash –x miscript.sh**

Variables

BASH no es fuertemente tipado, las variables pueden cambiar su tipo durante la ejecución de nuestros scripts.
Las variables tienen básicamente dos tipos: String o array. No precisan declaración, se las utiliza directamente.
Los nombres de las variables pueden contener mayúsculas, minúsculas, números y el símbolo "_", pero **NO pueden comenzar con un número**.

```
NOMBRE="Fulano De Tal"
facultad=Informatica
carrera_1="Licenciatura en Sistemas"
carrera_2="Licenciatura en Informatica"
echo El alumno $NOMBRE de la Facultad de $facultad
cursa $carrera_1 y $carrera_2
# imprime:
#  El alumno Fulano De Tal de la Facultad de
#  Informática cursa Licenciatura en Sistemas
#  y Licenciatura en Informatica
```

(*) Notar que en la asignación no se deben dejar espacios.

```
nombre=Carlos
echo "Hola $nombre" # Hola Carlos
echo Hola ${nombre} # Hola Carlos
```

Variables: Arreglos

Los índices en los arreglos comienzan en 0.

Creación arreglo vacío	arreglo_a=()
Creación arreglo inicializado	areglo_b=(1 3 3 555 6)
Asignación de un valor en una posición del arreglo	arreglo_b[2]=valor
Acceso a un valor del arreglo (llaves obligatorias)	${arreglo_b[2]}
Acceso a todos los valores (@ o *)	${arreglo[@]} o ${arreglo[*]}
Tamaño del arreglo	${#arreglo[@]} o ${#arreglo[*]}
Borrado de un elemento (no elimina la posición, solo deja las posición vacía)	unset arreglo[2]

Uso de comillas y expansión de parámetros

Se puede omitir las comillas al ingresar valores string cuando:

- El string se compone de una sola palabra.
- En el parámetro echo.
- Sabemos que el valor de una variable no tiene espacios o es un número.

Comillas dobles: Los string son evaluados y se reemplazan en ellos las expresiones o se expanden variables.
Comillas simples: Se toman como literales, no se efectúa ningún reemplazo.

```
nombre=Pepe
echo “$nombre”        #imprime Pepe
echo ‘$nombre’        #imprime $nombre
```

Concatenación:
Variable3=”Así concateno $variable1 a otro string.”

Sustitución de comandos
Permite utilizar la salida de un comando como si fuese una cadena de texto normal, permite guardarlo en variables o utilizarlos directamente. Se la puede utilizar de dos formas:

- $(comando)
- `comando`

Ejemplo:

```
arch=”$(ls)”
arch=”`ls`”
arch=`ls`
mis_arhivos=”$(ls /home/$(whoami))”
```

Evaluacion de condiciones logicas
Las condiciones lógicas normalmente se evalúan mediante:

- [condicion]
- Test condición

Operadores para condición:

Operador	Entre Variables String	Entre Números
Igualdad	$nombre == “Maria”	$nombre -eq “Maria”
Desigualdad	$nombre != “Maria”	$nombre -ne “Maria”
Mayor	A > Z	2 -gt 3
Mayor o igual	A >= Z	2 -ge 3
Menor	A < Z	2 -lt 3
Menor o igual	A <= Z	2 -le 3
Negación	! expresion	! expresion

Argumentos y valor de retorno
Los scripts pueden recibir argumentos en su invocación, para accederlos se utilizan variables especiales:

- $0 contiene la invocación al script
- $1, $2… contienen cada uno de los argumentos
- $# contiene la cantidad de argumentos recibidos
- $* contiene la lista de todos los argumentos
- $? Contiene en todo momento el valor de retorno del último comando ejecutado

Función exit

Causa la terminación de un script, puede devolver cualquier valor entre 0 y 255, donde el valor 0 indica que el script se ejecutó con éxito y un valor distinto indica error.
Se puede consultar el exit status imprimiendo la variable $?.

Funciones

Se pueden declarar de 2 formas:

- function nombre { block }
- nombre () { block }

Con la sentencia **return** se retorna un valor entre 0 y 255, y se puede consultar con la variable $?. También reciben argumentos.

```
# recibe 2 argumentos y retorna:
#  1 si el primero es el mayor
#  0 en caso contrario
mayor() {
  echo "Se van a comparar los valores $*"
  if [ $1 -gt $2 ]; then
    echo "$1 es el mayor"
    return 1
  fi
  echo "$2 es el mayor"
  return 0
}
mayor 5 6 # Invocación
echo $?   # Imprime el resultado
```

Variables: alcance y visibilidad

Las variables no inicializadas son reemplazadas por nulo o 0 según el contexto de evaluación. Por defecto las variables son globales. Una variable local en una función se define con **local nom_var.**

Estructuras de control

- Decisión:

```
if [ condition ]
then
  block
fi
```

- Selección:

```
case $variable in
  "valor 1")
    block
    ;;
  "valor 2")
    block
    ;;
  *)
    block
    ;;
esac
```

- **Menú de opciones:**
  - `select variable in alternativa1 alternativa2 alternativa3`
    `do`
    `  # en $variable está el valor elegido`
    `  block`
    `done`

Ejemplo:

```
select action in new exit
do
 case $action in
  "new")
   echo "Selected option is NEW"
   ;;
  "exit")
   exit 0
   ;;
 esac
done
```

→

```
1) new
2) exit
#?
```

- **Iteración - bloques FOR:**
  - **Al estilo de C:**

```
for ((i=0; i < 10; i++))
do
  block
done
```

  - **Con lista de valores** (notar diferencia con Pascal!!!):

```
for i in value1 value2 value3 valueN;
do
  block
done
```

- **Iteración - Bloques WHILE y REPEAT (until):**
  - `while [ condition ] #Mientras se cumpla`
    `# la condición`

```
do
  block
done
```

  - `until [ condition ]#Mientras no se`
    `#cumpla la condición`

```
do
  block
done
```

```
if [ "$USER" == root ]
then
    echo "super user"
else
    echo "Ud. es $USER"
fi
```

```
n=0
while [ $n -ne 5 ]; do
  echo $n
  let n++
done
```

```
for archivo in $(ls)
do
  echo "- $archivo"
done
```

```
for ((i=0; i<5; i++))
do
  echo $i
done
```

- Adicionalmente:
  - `break` ***[n]*** corta la ejecución de ***n*** niveles de bucles
  - `continue` ***[n]*** salta a la siguiente iteración del enésimo loop que contiene esta instrucción

```
#!/bin/bash
# Imprime los numeros del 1 al 5
# (no es un código para nada elegante)
i=0
# true es un comando que siempre retorna 0
while true
do
  let i++   # Incrementa i en 1
  if [ $i -eq 6 ]; then
    break   # Corta el loop (while)
  fi
  echo $i
done
```

```
i=0
while true
do
    let i++
    if [ $i -eq 6 ]; then
        break    # Corta el while
    elif [ $i -eq 3 ]; then
        continue # Salta una iteración
    fi
    echo $i
done
```