

Uso de máquinas virtuales y shell de comandos

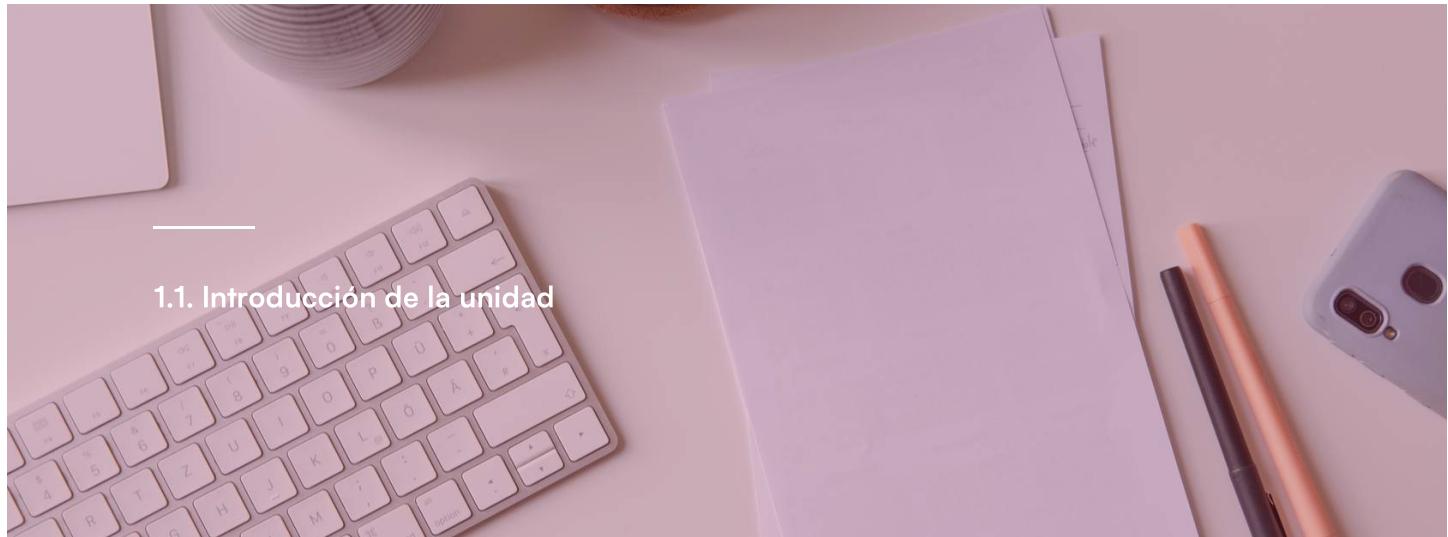


- ☰ I. Introducción
- ☰ II. Objetivos
- ☰ III. Concepto de máquina virtual. Virtual Box
- ☰ IV. Creación y configuración de una máquina virtual
- ☰ V. Carga de una máquina virtual
- ☰ VI. La shell de comandos de Linux. Creación de scripts
- ☰ VII. Resumen
- ☰ VIII. Caso práctico con solución
- ☰ IX. Lecturas recomendadas
- ☰ X. Glosario
- ☰ XI. Bibliografía

QUESTION BANKS

I. Introducción

1.1. Introducción de la unidad



Una de las principales dificultades que surge en el ámbito del desarrollo de software es la **incompatibilidad de las herramientas que se van a utilizar con respecto a los diferentes sistemas operativos**. Así, en muchas ocasiones, el **software no es multiplataforma**, es decir, se ha creado para ejecutarse en un sistema operativo concreto. Esto **genera problemas si el sistema operativo sobre el que se va a trabajar no coincide con el que precisa el software que se quiere utilizar**, por lo que en estos casos sería necesario instalar el sistema operativo requerido por el software.

Otro problema más particular, y ligado al ámbito del análisis de datos y del *big data*, es la **complejidad de la instalación y configuración de las herramientas informáticas que se utilizan**. En muchos casos, el proceso de instalar y configurar la herramienta con los parámetros adecuados **requiere conocimientos propios de un perfil administrador** muy diferente al del usuario de la herramienta concreta. A estas dificultades habría que añadir el tiempo requerido para ejecutar estas operaciones.

Asimismo, y dentro de este ámbito, los sistemas operativos basados en UNIX, como Linux y FreeBSD, ofrecen de manera natural un extenso abanico de herramientas para el procesamiento y filtrado de textos. Todas estas herramientas están disponibles en lo que se conoce como *shell*, que por definición es una aplicación capaz de procesar comandos textuales introducidos por el usuario y producir un resultado también en formato textual.

Todo esto, unido a la versatilidad del lenguaje de scripting que ofrecen las shells de Linux, proporciona al analista de datos y al científico de datos el laboratorio perfecto para procesar, filtrar y extraer información útil a partir de datos crudos que se puedan recoger de ficheros de texto, bases de datos, API públicas y privadas, etc.

En esta unidad del módulo se van a estudiar las máquinas virtuales como solución a estos problemas. Más concretamente, se analizará la herramienta **Virtual Box para crear y gestionar máquinas virtuales**. Asimismo, se introducirá el uso de la *shell* de comandos de Linux, ya que muchas de las herramientas que se utilizan en el ámbito del *big data* se ejecutan sobre entornos de tipo Linux y hay que interactuar con dichas herramientas mediante el uso de comandos de la *shell*.

CONTINUAR

La unidad se estructura de la siguiente forma.

En primer lugar, una introducción al concepto de máquina virtual y la necesidad de su uso.

A continuación, se describe cómo instalar Virtual Box en un entorno Windows y cómo se puede crear y configurar de manera básica una máquina virtual.

En la siguiente sección se mostrará cómo **cargar y utilizar una máquina virtual previamente creada**.

Después, y ya con la posibilidad de ejecutar un sistema operativo Linux desde la máquina virtual, **el terminal de comandos se convertirá en el espacio de trabajo**. Se muestran los **comandos básicos de la shell** que permiten al usuario moverse cómodamente por el sistema de ficheros.

Posteriormente se explican los **comandos más utilizados en el ámbito del análisis de datos** para procesar y filtrar ficheros de texto.

La unidad continúa desarrollando los **sistemas de automatización y las estructuras de control de flujo** que ofrece la *shell*.

Por último, se enseña a **componer guiones o scripts que contengan algoritmos de cierta complejidad** para la automatización de tareas.



Para obtener el máximo aprovechamiento de la unidad, es recomendable descargar **Oracle VM VirtualBox** y proceder a la **instalación y configuración de una máquina virtual** (Linux). Sirviéndose de esta instalación, es importante **desarrollar los guiones del último capítulo de la unidad**.

II. Objetivos



2.1. Objetivos de la unidad

Los objetivos que los alumnos alcanzarán tras el estudio de esta unidad son:

- 1 Entender el concepto y la necesidad del uso de máquinas virtuales.
- 2 Saber instalar la herramienta Virtual Box.
- 3 Saber crear y configurar de manera básica una máquina virtual en Virtual Box.
- 4 Saber cómo se carga una máquina virtual dada en Virtual Box.
- 5 Conocer y saber utilizar de manera básica la *shell* de comandos de Linux.

III. Concepto de máquina virtual. Virtual Box

La existencia de diferentes sistemas operativos acarrea, como una de las principales desventajas en el ámbito del desarrollo de software, la necesidad de tener que crear diferentes versiones para un mismo software, adecuadas para cada uno de los sistemas operativos.

Sin embargo, esto no se hace siempre, ya que la creación de una versión del mismo software para cada una de las diferentes plataformas **supone un esfuerzo económico para las empresas que no siempre es rentable**. Esta situación genera problemas en el momento en que se quiere utilizar un determinado software que solo se ejecuta para un determinado sistema operativo que no coincide con el que tiene el usuario.

Saber más

Antes de la aparición de las máquinas virtuales, la única solución posible consistía en instalar en la misma máquina los sistemas operativos que eran necesarios para el software que se iba a utilizar, de manera que compartían entre ellos el disco duro. Se trata de una solución costosa tanto en tiempo —el necesario para instalar y poner a punto el sistema operativo correspondiente— como en la cantidad de memoria del disco que deben compartir los diferentes sistemas operativos instalados.

En este sentido, las máquinas virtuales aparecen como una solución a la necesidad de tener que usar diferentes sistemas operativos en una misma máquina. La idea del funcionamiento de las máquinas virtuales consiste en **usar el sistema operativo instalado por defecto en la máquina**, denominado sistema anfitrión, **para ejecutar una simulación de otro sistema operativo denominado sistema huésped**. El sistema huésped se ejecuta como si fuera un programa más y se puede utilizar como si realmente estuviera instalado en la máquina —aunque realmente no se encuentra instalado y lo único que se está ejecutando es una simulación—.



Para poder crear y ejecutar una máquina virtual, es necesario un software especial encargado de llevar a cabo la simulación. Actualmente, las herramientas más utilizadas para la creación y gestión de máquinas virtuales son Virtual Box y VMware, si bien es cierto que existen otras muchas alternativas, algunas de ellas de uso comercial (en la siguiente nota al pie se añade un enlace a una lista con los sistemas de virtualización más usados en la actualidad¹).

¹[Top 10 Most Popular Virtualization Software April 2021](#).

CONTINUAR

VMWARE

VIRTUAL BOX

La herramienta **VMware** dispone de una **versión gratuita con las características y funciones necesarias para un usuario particular** (como la creación de una máquina virtual) y una **versión de pago orientada a empresas que añade funciones y características más avanzadas**, como la posibilidad de virtualizar sistemas operativos y gestionarlos a través de la red como si se tratase de una nube.

VMWARE

VIRTUAL BOX

Por otro lado, **Virtual Box** es una **herramienta de código abierto, totalmente gratuita y disponible para cualquier sistema operativo**, propiedad de Oracle. No ofrece las mismas características avanzadas orientadas hacia las empresas que sí tiene VMware, pero **cuenta con todas las funciones necesarias para crear y ejecutar máquinas virtuales e incluso dispone de algunas funcionalidades propias** como la capacidad de ejecutar varias máquinas virtuales a la vez.

Con respecto a la eficiencia y el rendimiento, **VMware presenta un rendimiento mejor que Virtual Box en el uso de la memoria y de la CPU**. Sin embargo, con **respecto al rendimiento del disco duro ambas herramientas son similares**.

Cuando se utilizan máquinas virtuales, hay que tener en cuenta lo siguiente:

Todas las operaciones que se realicen sobre y desde el sistema operativo virtualizado **no afectarán al sistema operativo anfitrión ni a los demás sistemas operativos huéspedes**. De hecho, esta es una de las ventajas de usar máquinas virtuales: cada sistema operativo virtualizado es independiente de los demás; si ocurre algún problema en el sistema operativo virtualizado, el sistema operativo anfitrión quedará resguardado y bastará con interrumpir la ejecución de la máquina virtual.

Cada vez que se ejecuta una máquina virtual, en el momento en que se apaga, es posible guardar el estado de la máquina virtual.

Es decir, es posible mantener el sistema tal como estaba antes de apagarse, de manera que, cuando se vuelva a ejecutar, el usuario podrá continuar en el mismo punto en que lo dejó.

En general, **es posible el intercambio de documentos entre el sistema operativo anfitrión y el huésped**, lo que facilita la reutilización de todo lo generado en un sistema operativo virtualizado y el uso de recursos que se encuentran en el sistema operativo anfitrión.

Desde un sistema operativo virtualizado **se pueden utilizar recursos auxiliares como el acceso a la red de comunicación, impresoras u otros dispositivos** que se encuentran conectados a la máquina sobre la que se ejecuta.

A cada máquina virtual se le asignan recursos según los disponibles del sistema operativo anfitrión, tales como memoria RAM, capacidad de almacenamiento y núcleos de CPU.

CONTINUAR

Hay dos casos de uso principales cuando se utilizan este tipo de herramientas: **crear una máquina virtual** o **cargar una máquina virtual**.

Crear una máquina virtual

El primer caso se presenta **cuando surge la necesidad de preparar un entorno virtual para ejecutar determinado software y no es posible hacerlo en el sistema operativo instalado por defecto** en la máquina.

Cargar una máquina virtual

El otro caso de uso consiste en **utilizar una máquina virtual que alguien ha creado y configurado previamente y que se desea utilizar para poder acceder a determinado software instalado en ella**. Obsérvese, con respecto a este último caso, que **existen sitios web que actúan como repositorios de máquinas virtuales creadas por otros usuarios** que las ponen a libre disposición de otros para que puedan descargarlas y usarlas.



También es preciso llamar la atención sobre el hecho de que, además de las herramientas comentadas anteriormente, existe otra forma alternativa de crear y utilizar máquinas virtuales en el contexto de los denominados servicios en la nube.

La computación en la nube –también denominada *cloud computing*– es un **modelo de prestación de servicios en el que los usuarios pueden acceder a servicios o recursos que se encuentran en internet mediante el pago por el consumo efectuado y, en algunos casos, de forma gratuita**. Entre los servicios ofrecidos en la nube por parte de las empresas que se dedican a ello, se encuentra la **creación de máquinas virtuales**, también denominadas **instancias**.

La creación de una instancia consiste en **definir las características de la máquina virtual**: sistema operativo, capacidad de almacenamiento, CPU, memoria, capacidad de red... A la instancia **se asocia una IP pública que servirá para la conexión**. Existe una relación entre las herramientas de virtualización y las instancias de máquinas virtuales de los servicios de la nube que consiste en la posibilidad de importar y exportar máquinas virtuales de un servicio en la nube a una máquina virtual local y viceversa.

Ejemplo

En la página web de Amazon se describe la importación y exportación de máquinas virtuales entre los servicios en la nube de Amazon (AWS) y la herramienta de virtualización VMware².

²[Amazon. VM Import/Export](#).

Esta unidad se va a centrar en el **uso de Virtual Box**, ya que cubre las necesidades esenciales requeridas en este ámbito, que son esencialmente **evitar las incompatibilidades del software que se va a utilizar con el sistema operativo instalado en una máquina** —para asegurar que se podrá hacer uso del software con independencia de la máquina utilizada— y **evitar al usuario el tener que instalar y configurar el software al usar para ello máquinas virtuales que ya lo tienen instalado y configurado**, por lo que solo necesitará cargar la máquina virtual correspondiente.

IV. Creación y configuración de una máquina virtual

Virtual Box es un programa de software libre que permite la instalación de otro sistema operativo, dentro del sistema existente, y la interacción con él.

De esta forma, **es posible instalar programas diseñados para los sistemas operativos clientes**. En esta sección se describe **cómo instalar Virtual Box**, además de cómo se puede **crear y configurar de manera básica una máquina virtual**.

4.1. Requerimientos para la instalación de Virtual Box

Los requisitos mínimos para poder ejecutar Virtual Box son relativamente reducidos³:

- Al menos 512 Mb de memoria RAM.
- Almacenamiento. La instalación de Virtual Box ocupa tan solo 30 Mb, pero habrá que contar con almacenamiento extra para poder albergar las máquinas virtuales huésped (considerando que una instalación de Windows 10 puede ocupar de inicio unos 15 Gb, habría que disponer al menos de ese espacio libre en el sistema anfitrión).
- Procesador con arquitectura x86 (AMD o Intel).

³[VirtualBox End User Documentation](#).

No obstante, para garantizar un funcionamiento óptimo tanto del sistema huésped como del sistema anfitrión, las **recomendaciones** serían las siguientes:

MEMORIA RAM	ALMACENAMIENTO	PROCESADOR
<p>Mínimo 8 GB, ya que garantizará que el sistema operativo anfitrión y el huésped cuenten con suficiente memoria.</p>		

MEMORIA RAM	ALMACENAMIENTO	PROCESADOR
-------------	----------------	------------

Al menos **20 GB libres** para asignar al sistema operativo huésped.

MEMORIA RAM	ALMACENAMIENTO	PROCESADOR
-------------	----------------	------------

El ordenador debe contar con **un procesador relativamente nuevo**, Intel o AMD, ya que el sistema operativo anfitrión comparte recursos con el sistema operativo huésped y, además, el procesador debe soportar la virtualización; esta función debe habilitarse desde el BIOS. En la página web de BlueStacks se describe el proceso para verificar si un procesador soporta la virtualización y cómo habilitarla⁴.

⁴[BlueStacks: "¿Cómo puedo habilitar la virtualización \(VT\) en mi PC?"](#)

CONTINUAR

4.2. Instalación de Virtual Box

Para instalar Virtual Box, en primer lugar hay que descargar el software desde la siguiente dirección (figura 1): [sitio de descargas de Virtual Box](#).

En esta unidad se enseña **cómo descargar e instalar la versión 6.1.22**. Sin embargo, es aconsejable descargar la última versión que esté disponible. En las siguientes explicaciones, se utilizará la versión para Windows y se usarán funciones compatibles entre distintas versiones de Virtual Box (crear, importar y exportar). Para el resto de los sistemas operativos el proceso es similar.



Figura 1. Sitio de descargas de Virtual Box.

Fuente: elaboración propia.

En la página de descargas, se pulsa sobre el enlace correspondiente al sistema operativo sobre el que se quiere instalar —en este caso sobre Windows hosts— y comienza la descarga del fichero “VirtualBox-<version>-Win.exe”.



Figura 2. Instalador de Virtual Box.

Fuente: elaboración propia.

Una vez que se ha bajado el archivo ejecutable, se pulsa sobre el instalador para que comience la ejecución del asistente (figura 2).

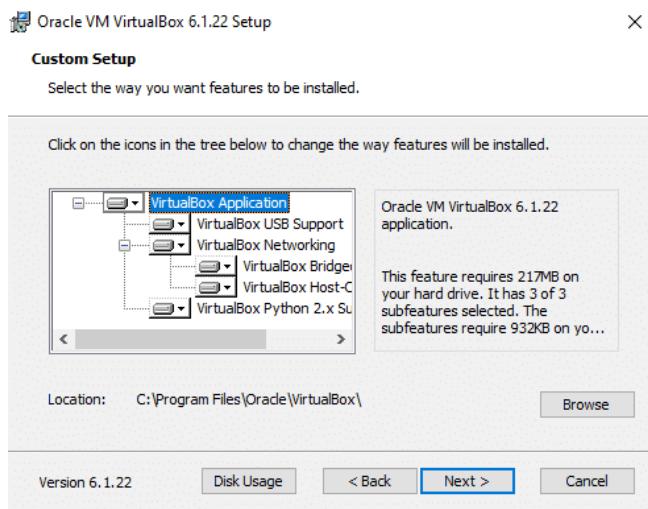


Figura 3. Ventana *Custom Setup*.

Fuente: elaboración propia.

Se pulsa sobre Next y aparece la ventana Custom Setup. Se deja la selección "VirtualBox Application" y se comprueba que la carpeta de instalación elegida por el instalador es "C:\Program Files\Oracle\VirtualBox" (figura 3).

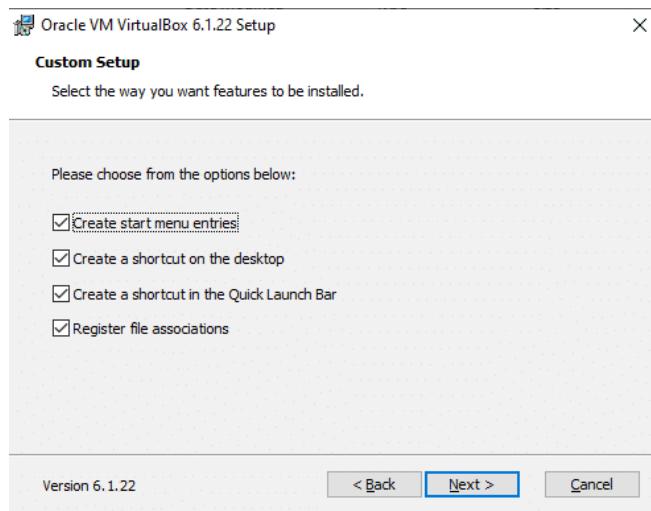


Figura 4. Opciones de configuración.

Fuente: elaboración propia.

Se pulsa sobre Next en la siguiente ventana, sin cambiar las opciones que aparecen por defecto (figura 4).



Figura 5. Ventana de desconexión de la red.

Fuente: elaboración propia.

En la siguiente ventana que aparece, se indica que el ordenador se va a desconectar momentáneamente de la red. Se pulsa sobre Yes (figura 5).

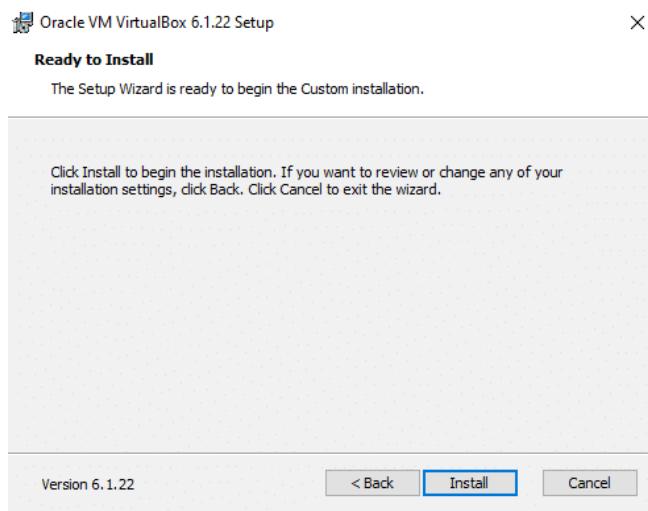


Figura 6. Ventana de instalación.

Fuente: elaboración propia.

Por último, en la siguiente ventana se pulsa sobre *Install* para que comience la instalación (figura 6).

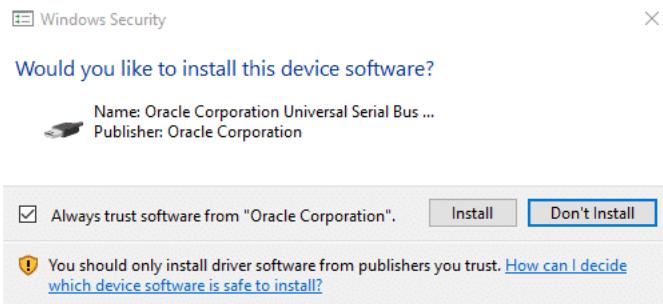


Figura 7. Instalar controlador USB.

Fuente: elaboración propia.

A continuación, aparecerá un aviso de Windows para preguntar si el programa puede hacer cambios en el sistema. Se pulsa sobre Yes y se deja que el instalador se ejecute hasta que finalice la instalación.

Durante el proceso se solicitará la instalación de software adicional para el control de puertos USB. Hay que pulsar en *Install*, ya que este software permitirá conectar dispositivos USB en el sistema huésped (figura 7).



Figura 8. Ventana de finalización de instalación.

Fuente: elaboración propia.

Cuando aparece la pantalla de fin de instalación, se pulsa sobre *Finish*, acción que ejecutará Virtual Box (figura 8).

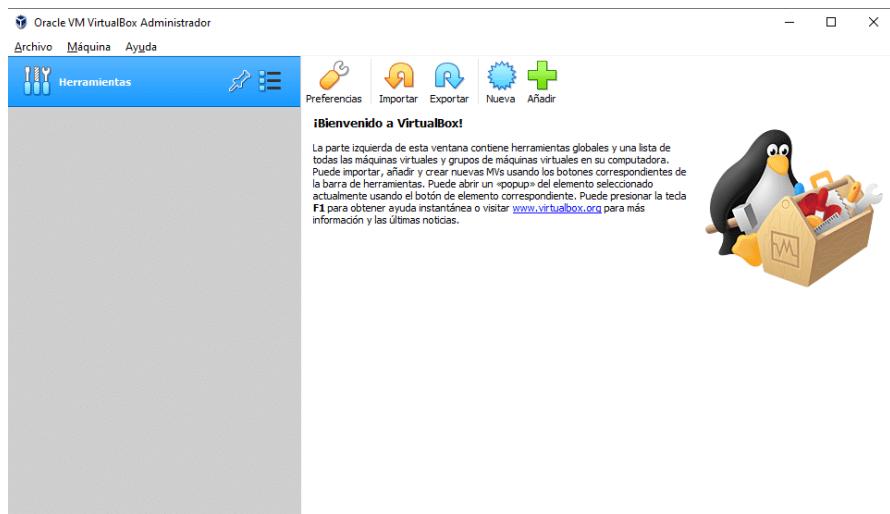


Figura 9. Interfaz principal de Virtual Box.

Fuente: elaboración propia.

En la figura 9 se puede ver la interfaz principal de Virtual Box.

CONTINUAR

4.3. Creación y configuración de una máquina virtual

Para crear una máquina virtual, lo primero que se necesita es una imagen del sistema operativo que se desea virtualizar.

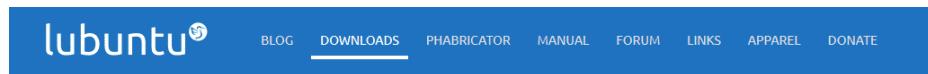


Figura 10. Página de descargas de Lubuntu.

Fuente: elaboración propia.

Para ilustrar la creación de la máquina virtual, se va a utilizar Lubuntu. Se trata de una distribución de Linux pensada para máquinas que disponen de pocos recursos software. La descarga de Lubuntu se realiza desde la [página web de Lubuntu](#). (figura 10).

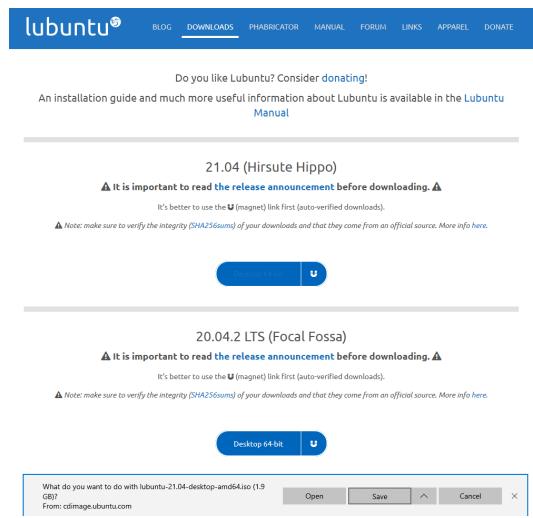


Figura 11. Descarga de Lubuntu.

Fuente: elaboración propia.

En la página de descargas, se pulsa sobre el enlace más adecuado para la máquina en cuestión (figura 11).

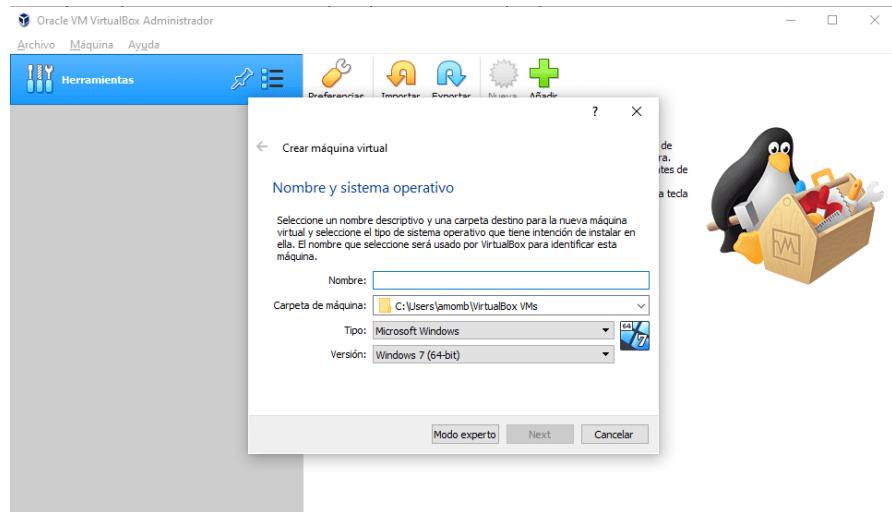


Figura 12. Asistente para crear una máquina virtual.

Fuente: elaboración propia.

Una vez descargado Lubuntu, se procede a la creación de la máquina virtual. Para ello, se abre Virtual Box pulsando sobre el ícono que habrá en el escritorio o bien en la lista de programas, o bien sobre el archivo "VirtualBox.exe" que se encuentra en la carpeta de instalación "C:\Program Files\Oracle\VirtualBox". En la interfaz principal de Virtual Box, se pulsa sobre el ícono *Nueva* para crear una nueva máquina virtual. Como resultado, se abre el asistente (figura 12).

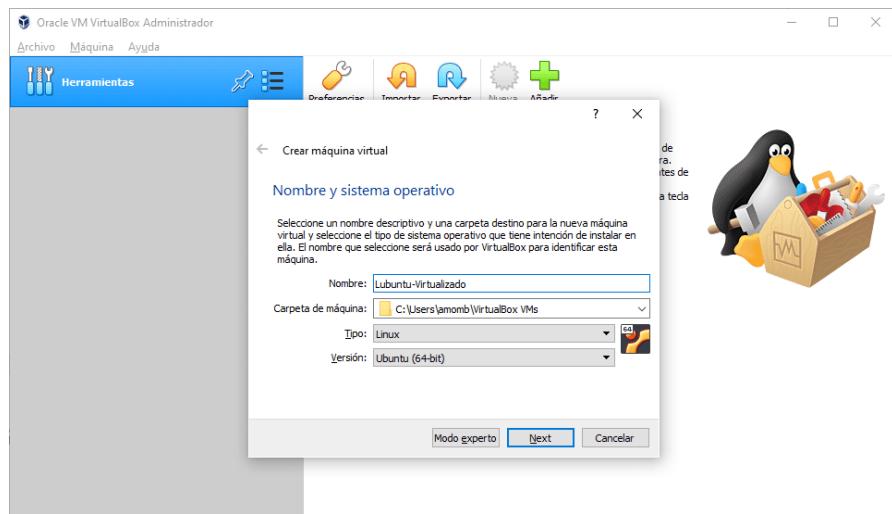


Figura 13. Datos de la máquina virtual.

Fuente: elaboración propia.

En el asistente, se inserta un nombre para la máquina virtual (por ejemplo, “Lubuntu-Virtualizado”), se elige como tipo “Linux” y como versión “Ubuntu (64-bit)”. Por último, se pulsa sobre *Next* (figura 13).

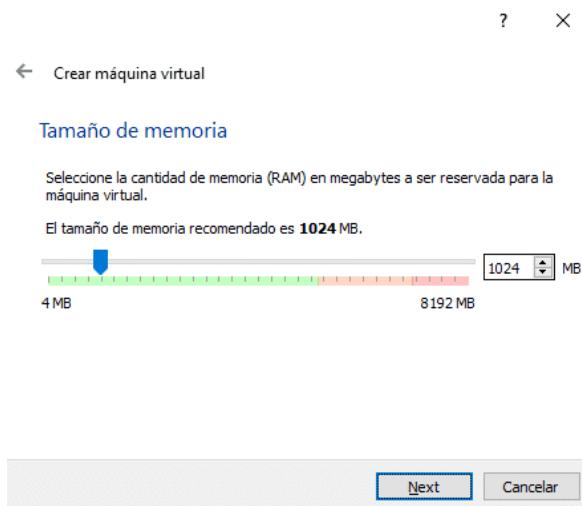


Figura 14. Tamaño de memoria de la máquina virtual.

Fuente: elaboración propia.

En la siguiente ventana, se debe elegir cuánta memoria se va a asignar al sistema virtualizado (memoria que se quita a Windows) y se pulsa sobre Next. Se puede seleccionar la recomendación que indica el asistente (figura 14).

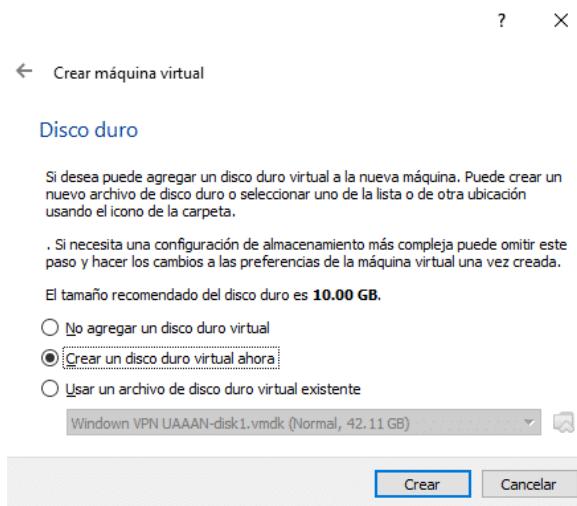


Figura 15. Creación de máquina virtual.

Fuente: elaboración propia.

En la siguiente ventana, se verifica que se ha seleccionado la opción “Crear disco virtual ahora” y se pulsa sobre *Crear* (figura 15).

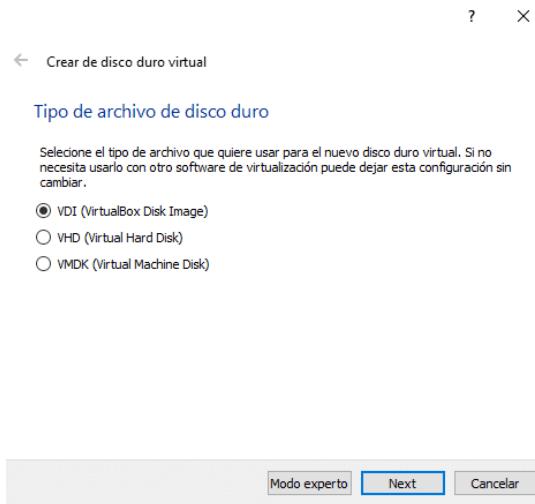


Figura 16. Selección del tipo de archivo del disco duro.

Fuente: elaboración propia.

En la siguiente ventana, se elige el tipo de archivo que se desea utilizar para el disco duro virtual. Se puede dejar la recomendación que indica el asistente (figura 16) y se pulsa sobre *Next*.

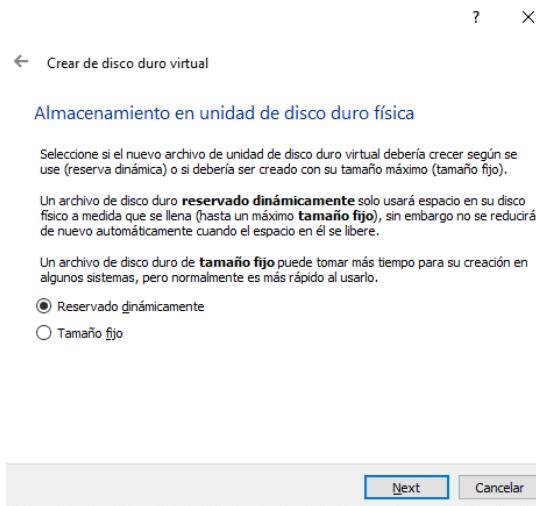


Figura 17. Selección del tipo de almacenamiento en el disco duro.

Fuente: elaboración propia.

En la siguiente ventana, se selecciona el tipo de almacenamiento en el disco duro. Se mantiene la recomendación que indica el asistente (figura 17) y se pulsa sobre *Next*.

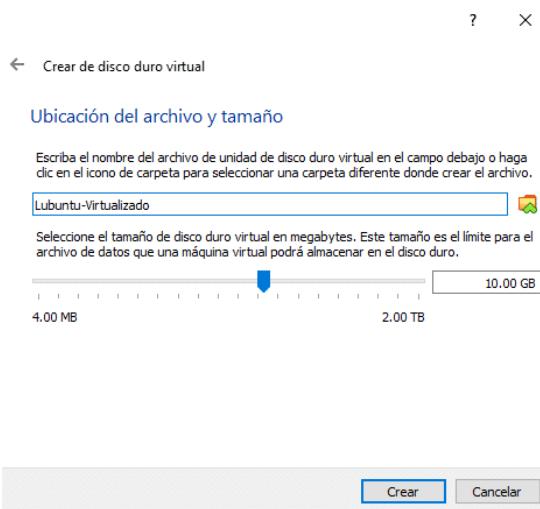


Figura 18. Ubicación del archivo del disco duro.

Fuente: elaboración propia.

En la siguiente ventana, se pregunta al usuario dónde se debe ubicar el archivo. Por defecto, se guardará en: "C:\Users\nombre_usuario\VirtualBox VMs\Nombre-Máquina Virtual".

Se puede cambiar pulsando sobre el ícono en forma de carpeta. Asimismo, en esta ventana se puede configurar el tamaño máximo del disco duro. Se puede dejar el que indica por defecto el asistente (figura 18).

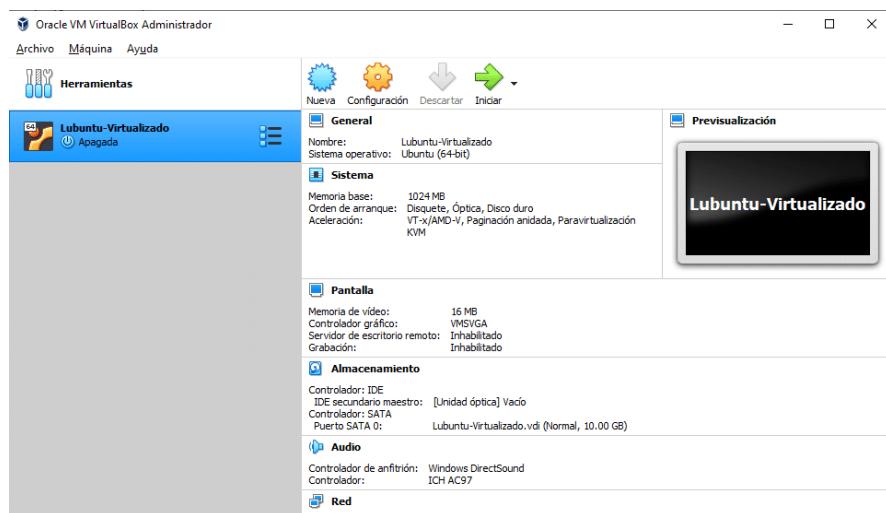


Figura 19. Máquina virtual creada.

Fuente: elaboración propia.

Al pulsar sobre *Crear*, aparece la ventana de Virtual Box actualizada con la máquina que se ha configurado (figura 19).

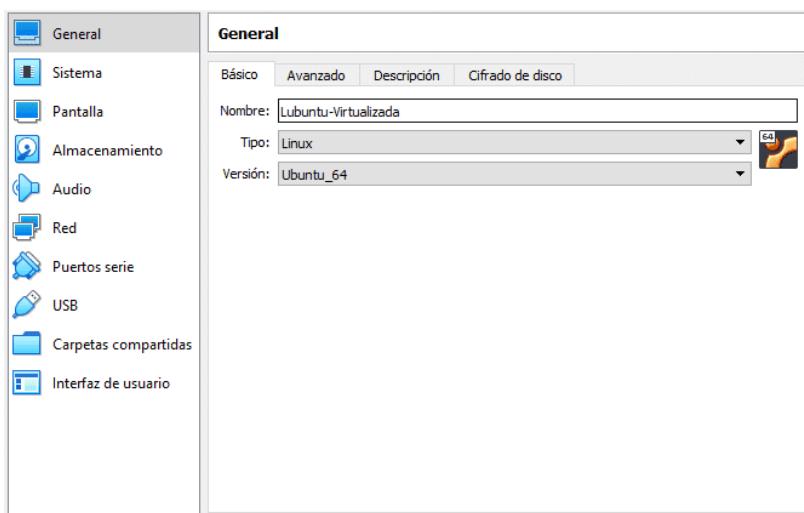


Figura 20. Configuración de la máquina virtual.

Fuente: elaboración propia.

La máquina virtual aún no se ha creado, lo que se ha hecho es configurar las características de la máquina. Antes de crear la máquina es necesario configurar algunas más, para lo cual hay que seleccionar la máquina y pulsar en la pestaña *Configuración* de la ventana mostrada en la figura 20.

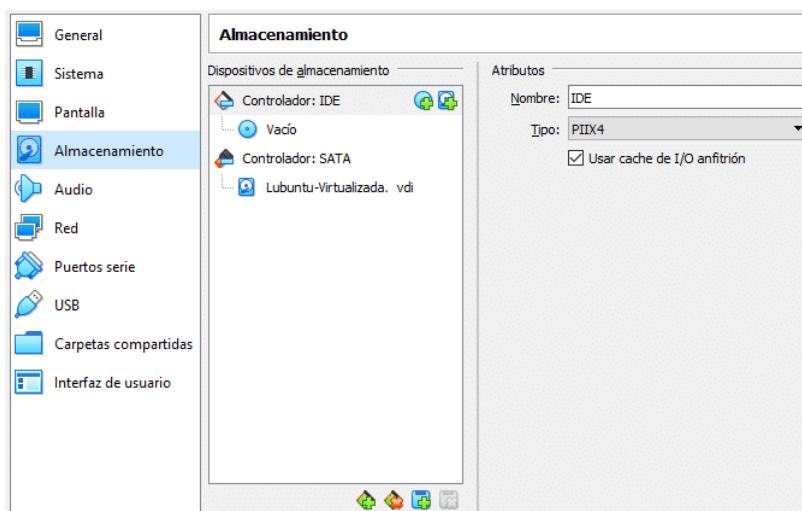


Figura 21. Pestaña Almacenamiento del asistente de configuración.

Fuente: elaboración propia.

En esta ventana, se pulsa sobre *Almacenamiento* y aparece la pestaña que se muestra en la figura 21.

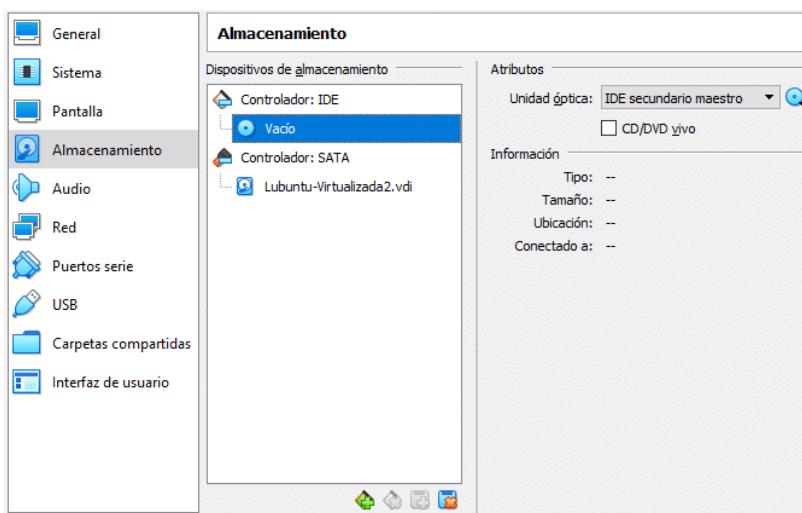


Figura 22. Configuración del controlador.

Fuente: elaboración propia.

En esta pestaña, se pulsa en el ícono del CD de “Controlador: IDE” en el cuadro del árbol de “Dispositivos de almacenamiento”, que aparece vacío (figura 22).

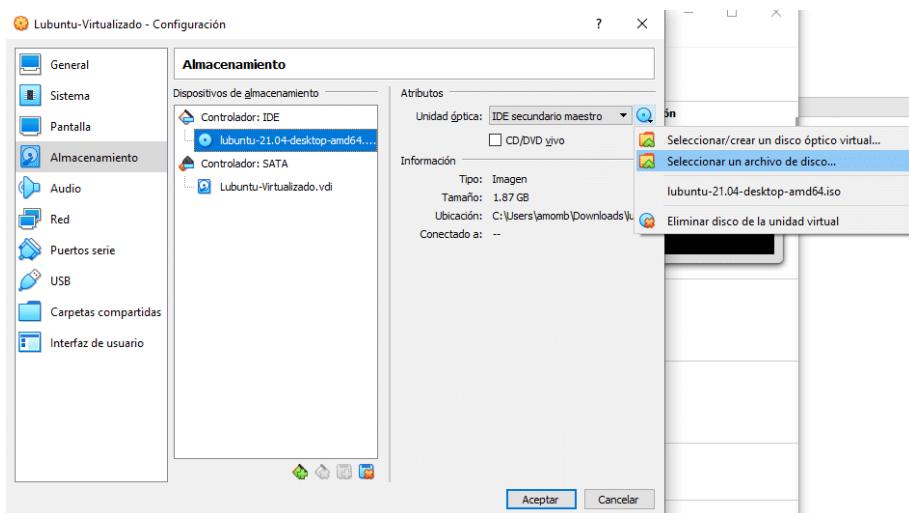


Figura 23. Selección del archivo.

Fuente: elaboración propia.

En el cuadro *Atributos*, se pulsa sobre el ícono del CD y aparece una lista desplegable. Se selecciona “Seleccione archivo de disco óptico virtual...” (figura 23).

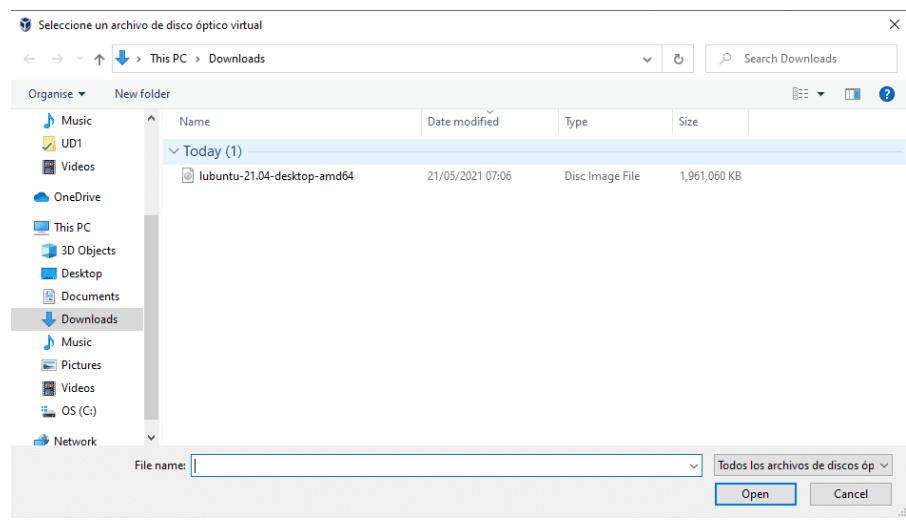


Figura 24. Selección del archivo de Lubuntu.

Fuente: elaboración propia.

Aparece un navegador de archivos y se selecciona el archivo de Lubuntu que se ha descargado previamente (figura 24).

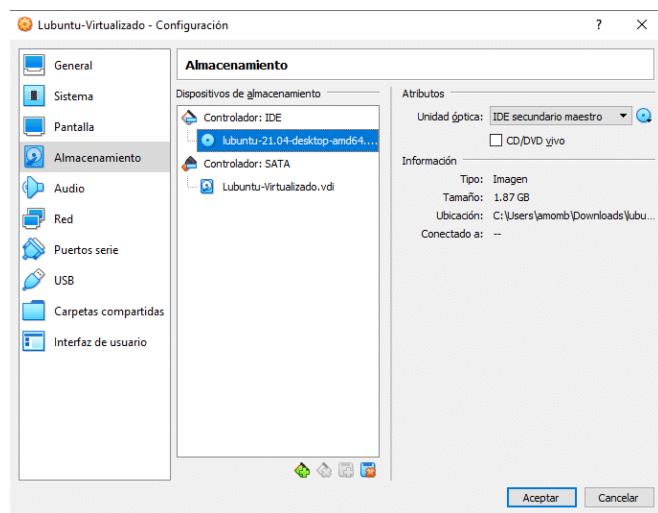


Figura 25. Pantalla resultante de la configuración.

Fuente: elaboración propia.

Tras esta selección, la pantalla se muestra como en la figura 25.

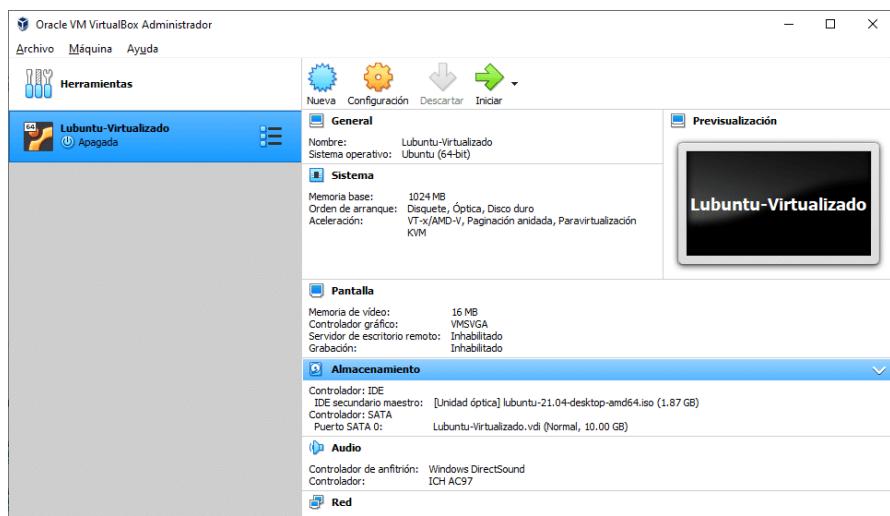


Figura 26. Interfaz principal actualizada.

Fuente: elaboración propia.

A continuación, se pulsa sobre *OK* para volver a la interfaz principal de Virtual Box, en la que se puede observar que se ha actualizado el cuadro *Almacenamiento* (figura 26).

CONTINUAR

Ahora se va a crear el sistema virtualizado.

Desde la pantalla principal, se pulsa sobre *Iniciar* para que se cargue el sistema. Aparecerán diferentes mensajes y pantallas de instalación. Son mensajes similares a los que se mostrarían si se estuviera instalando realmente el sistema operativo. **Estos mensajes solo aparecerán la primera vez que se cargue el sistema virtualizado**, no en las restantes ocasiones.

Recuerda:

Hay que tener en cuenta que, por defecto, Virtual Box activa la opción de **autocaptura del teclado**. Esto implica que la máquina virtual en ejecución “capturará” el teclado cada vez que su ventana esté activa. De esta manera, se enviará a la máquina virtual cada pulsación del teclado que se haga en ese momento. Para “liberar” el teclado y poder volver a utilizarlo en la máquina anfitriona, basta con pulsar la “tecla anfitrión”, normalmente **Ctrl+DERECHA**.

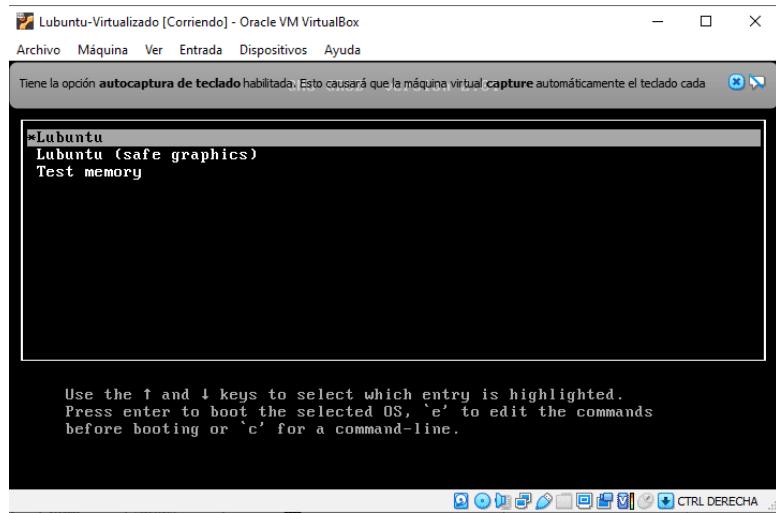


Figura 27. Pantalla de selección de operación.

Fuente: elaboración propia.

Se selecciona "Lubuntu" (figura 27).

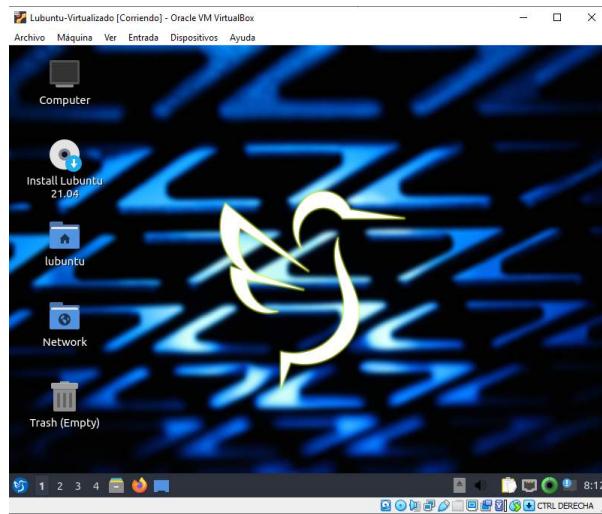


Figura 28. Inicio de la prueba de Lubuntu.

Fuente: elaboración propia.

Tras unos segundos, se iniciará el sistema operativo, en modo “prueba”, pero todavía no estará instalado en el disco duro virtual. Se hace doble clic en *Install Ubuntu <version>* (figura 28).

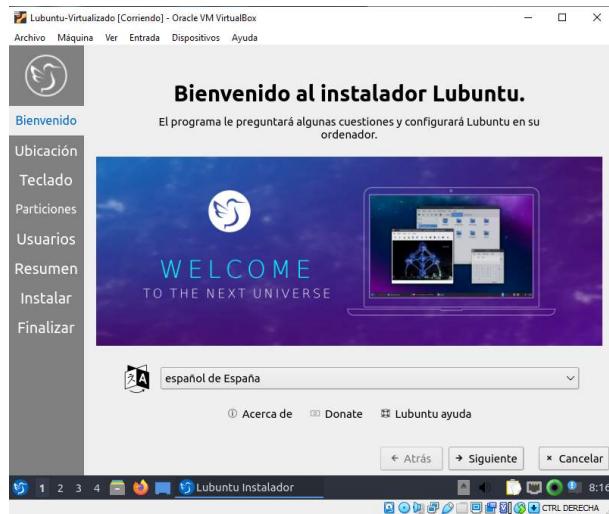


Figura 29. Instalador de Lubuntu.

Fuente: elaboración propia.

Se iniciará el instalador, cuyo primer paso es seleccionar el idioma, y luego se pulsa en *Siguiente* (figura 29).



Figura 30. Selección de la ubicación.

Fuente: elaboración propia.

Se selecciona la ubicación (figura 30).

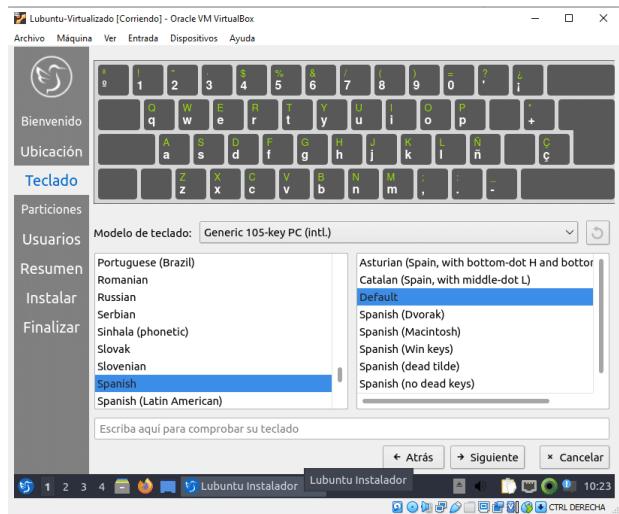


Figura 31. Selección del teclado.

Fuente: elaboración propia.

Se selecciona el tipo de teclado y se pulsa en *Siguiente* (figura 31).

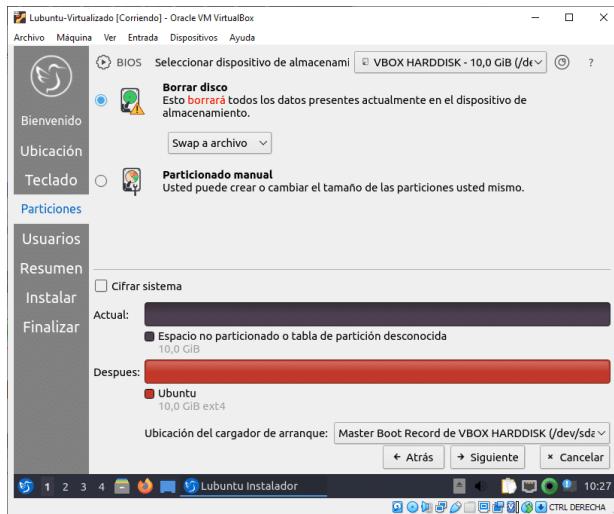


Figura 32. Selección de partición.

Fuente: elaboración propia.

En el siguiente paso se da la opción de particionar el disco duro virtual según las necesidades del usuario. En este ejemplo, se procede con la opción por defecto, que es utilizar una única partición, que borrará todo el disco virtual. Por tanto, se pulsa en *Siguiente* (figura 32).

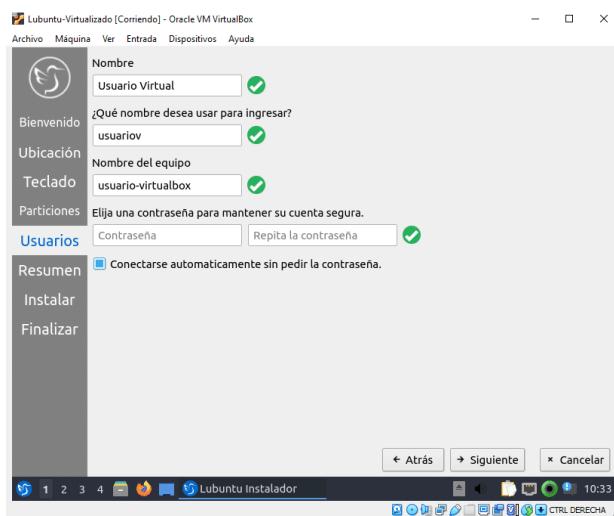


Figura 33. Creación del usuario administrador.

Fuente: elaboración propia.

Se crea un usuario que será el administrador del sistema. Si se desea, se puede definir una contraseña (figura 33).

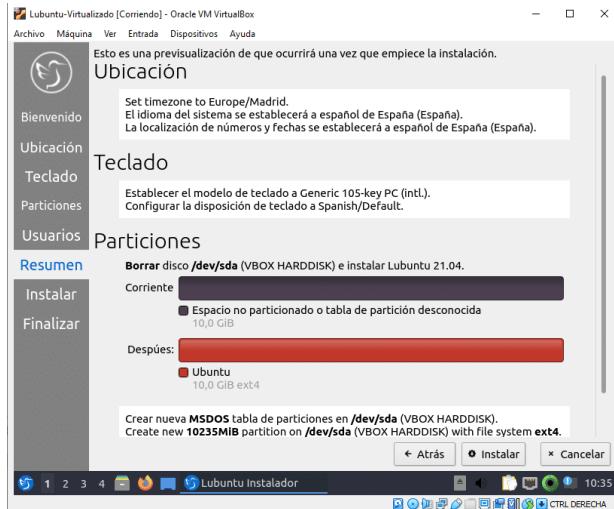


Figura 34. Previsualización de la instalación.

Fuente: elaboración propia.

A continuación, se muestra un **resumen** de lo que va a suceder durante el proceso de instalación. Se revisa que todo esté correcto y se pulsa en *Instalar* (figura 34).

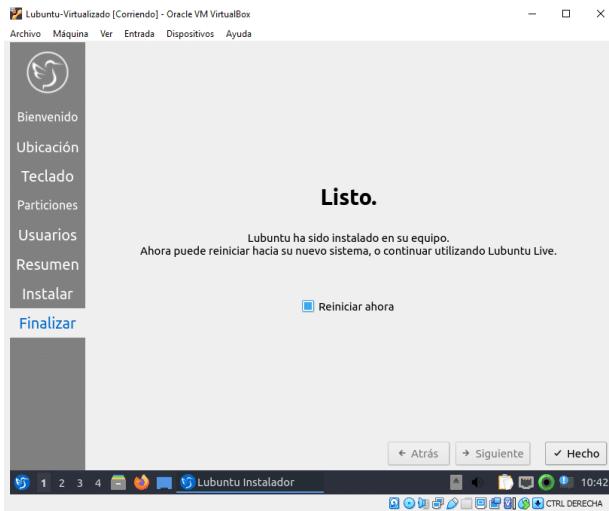


Figura 35. Instalación finalizada.

Fuente: elaboración propia.

Tras unos minutos, se habrá completado la instalación. Se pulsa en el botón *Hecho* para reiniciar el sistema (figura 35).

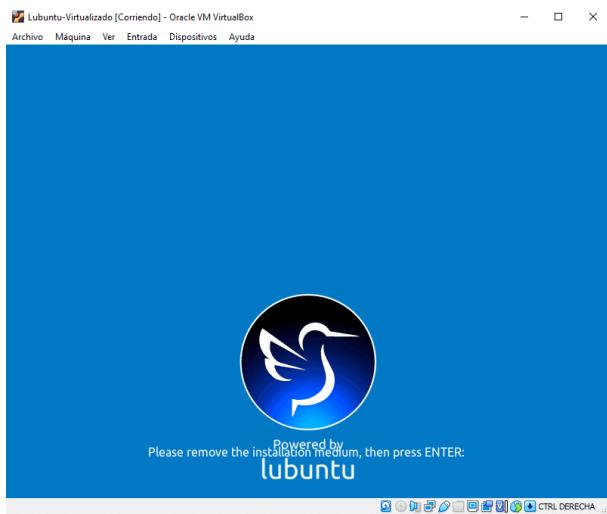


Figura 36. Instalación finalizada.

Fuente: elaboración propia.

Se pulsa la tecla *Enter* para reiniciar (figura 36).

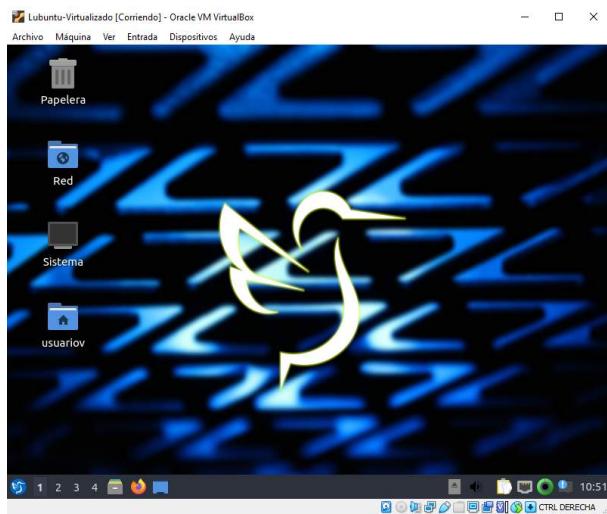


Figura 37. Escritorio.

Fuente: elaboración propia.

Una vez iniciado, se puede ver el escritorio de Lubuntu (figura 37).

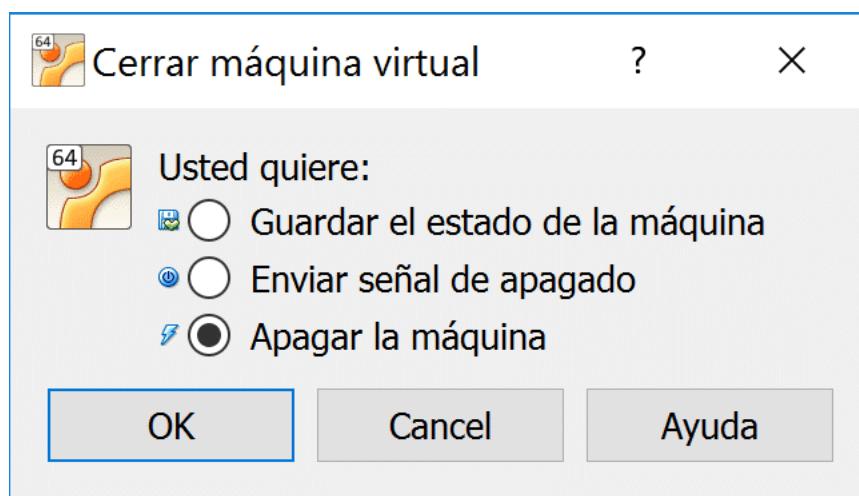


Figura 38. Opciones de cierre de la máquina virtual.

Fuente: elaboración propia.

Cuando se cierra la máquina virtual (Archivo > Cerrar), aparecen varias opciones (figura 38):

- **Guardar el estado de la máquina.** Cierra la MV y guarda el estado en el que se encuentra en ese momento, de modo que, al volver a iniciarla, continuará justo en el punto donde se cerró. No se pierde ningún tipo de información.
- **Enviar señal de apagado.** Es el apagado ACPI.
- **Apagar la máquina.** Equivale a cerrar la máquina sin guardar el estado, por lo que el SO no se cerrará correctamente y, al volver a iniciar la MV, el SO hará los chequeos previstos cuando este no se cierra correctamente. Es posible que se pierdan datos, por lo que debe usarse como último recurso cuando la MV se ha bloqueado.

La próxima vez que se quiera ejecutar el sistema virtual, bastará con seleccionar la máquina virtual y pulsar sobre *Iniciar* en la pantalla principal de Virtual Box.

V. Carga de una máquina virtual

Las máquinas virtuales son archivos que tienen el software de un sistema operativo virtualizado y permiten ejecutarlo en un sistema de virtualización. Virtual Box y VMWare tienen funciones que permiten importar y exportar máquinas virtuales previamente configuradas, con el fin de facilitar y reducir los tiempos de configuración.

Existen varios formatos para exportar/importar máquinas virtuales. Físicamente son archivos con extensión ***.ova**, ***.vdi**, etc. Los más comunes son ***.ova** y ***.vdi**, que son compatibles con Virtual Box y VMware.

OVA

VDI

El formato OVA (*Open Virtualization Application/Appliance*) es un **formato estándar para empaquetar servicios virtualizados**. Internamente es un fichero comprimido TAR que almacena una colección de ficheros necesarios para virtualizar uno o más sistemas operativos. Virtual Box soporta OVA desde su versión 2.2.0.

OVA

VDI

Por su parte, VDI (*Virtual Desktop Infrastructure*) es un **formato de imagen de un sistema operativo propio de Virtual Box**.

Hay repositorios gratuitos con máquinas virtuales ya preparadas para descargar e instalar:

- [OSBoxes](#), máquinas virtuales disponibles para VirtualBox y VMware de diferentes distribuciones Linux.
- [VirtualBoxes](#), máquinas virtuales disponibles para VirtualBox de diversos sistemas operativos libres o de código abierto.

The screenshot shows a web browser displaying the 'Virtual Images' website. The URL in the address bar is 'Home / Browse / System Administration / OS distribution / Virtual Images / Files'. The main header features a logo of a stylized water drop inside a diamond shape, followed by the text 'Virtual Images' and 'free virtual images for VirtualBox and VMWare'. Below this, it says 'Brought to you by: vincentzac'. A navigation menu at the top includes 'Summary', 'Files' (which is highlighted in blue), 'Reviews', and 'Support'. On the left, there's a sidebar with links like 'Home / Lubuntu', 'Parent folder', and 'Totals: 1 Item'. The main content area shows a table with one item: 'Lubuntu13-4.ova' (modified 2013-08-22, size 1.1 GB, downloads 1). To the right of the table are icons for a file download and an information symbol.

Figura 39. Descarga de una máquina virtual.

Fuente: elaboración propia.

Para ilustrar cómo cargar una máquina virtual, es posible descargar una de una distribución de Lubuntu desde el siguiente enlace (figura 39): [SourceForge](#).

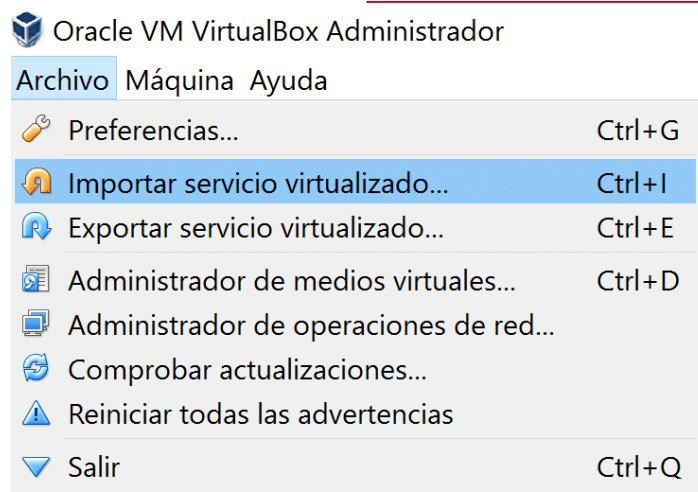


Figura 40. Importar máquina virtual.

Fuente: elaboración propia.

Una vez descargada, se importa en Virtual Box. Para ello, se selecciona en el menú la opción: *Archivos > Importar Servicio Virtualizado* y se selecciona la máquina virtual que se ha descargado (figura 40).

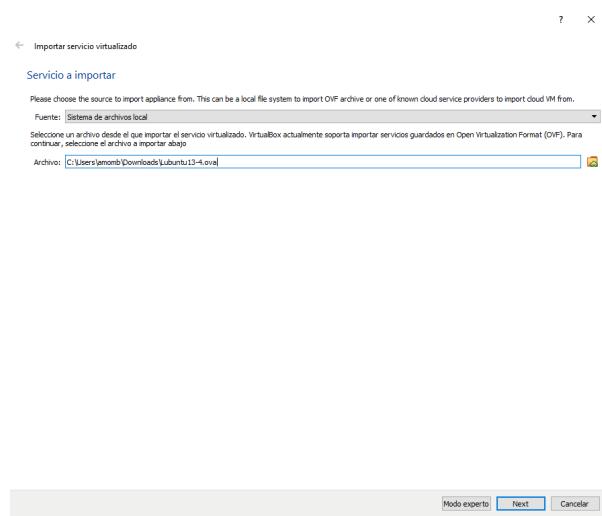


Figura 41. Selección de la máquina virtual que se quiere importar.

Fuente: elaboración propia.

A continuación, se selecciona la máquina virtual que se quiere importar (figura 41).

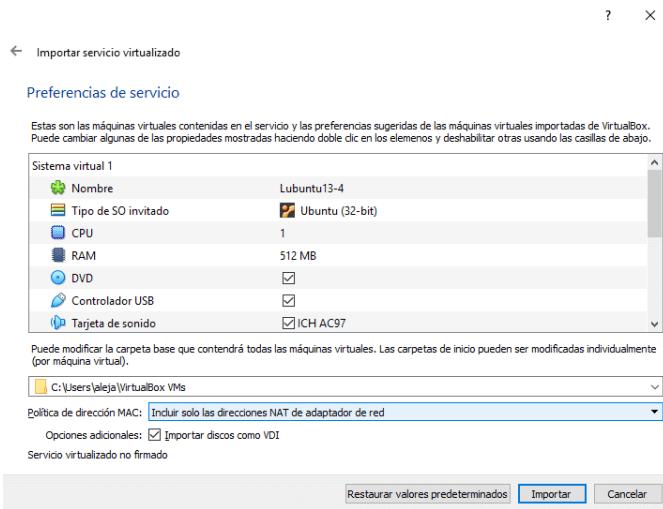


Figura 42. Configuración de la importación.

Fuente: elaboración propia.

A continuación, se pulsa en *Next* y aparecerá un resumen de la máquina virtual que se va a importar. Dependiendo de la versión de Virtual Box, se debe marcar la opción “Reiniciar la dirección MAC de todas las tarjetas de red” y pulsar en *Importar* (figura 42).

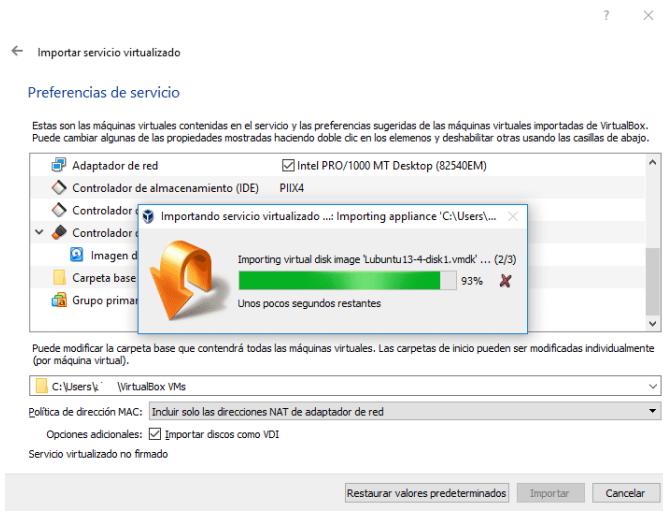


Figura 43. Importación de la máquina virtual.

Fuente: elaboración propia.

Se muestra una ventana con el estado del proceso de importación. El tiempo varía según las características de la máquina (figura 43).

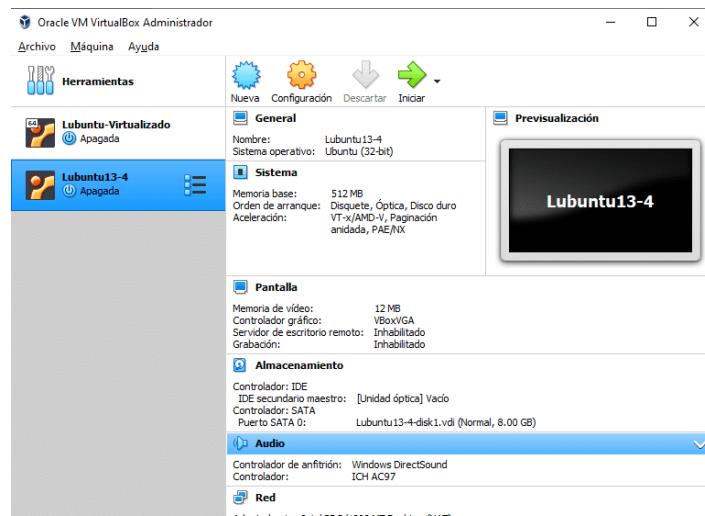


Figura 44. Máquina virtual importada.

Fuente: elaboración propia.

Una vez importada la máquina virtual, debería aparecer en la parte de la izquierda de la pantalla una nueva máquina virtual de nombre Lubuntu13-4 (figura 44).

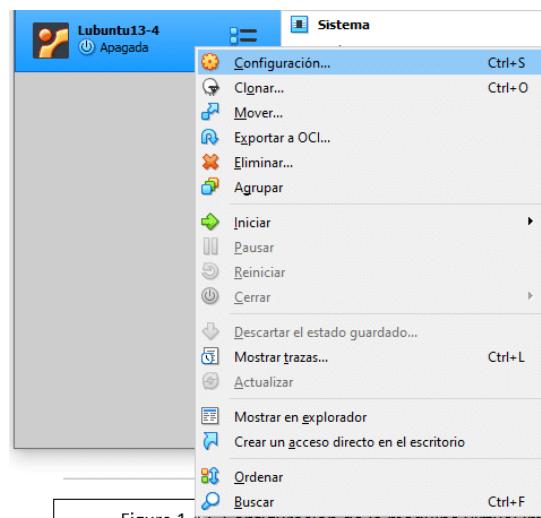


Figura 45. Configuración de la máquina virtual importada.

Fuente: elaboración propia.

Se pulsa sobre Lubuntu13-4 y se selecciona la opción *Configuración* (figura 45).

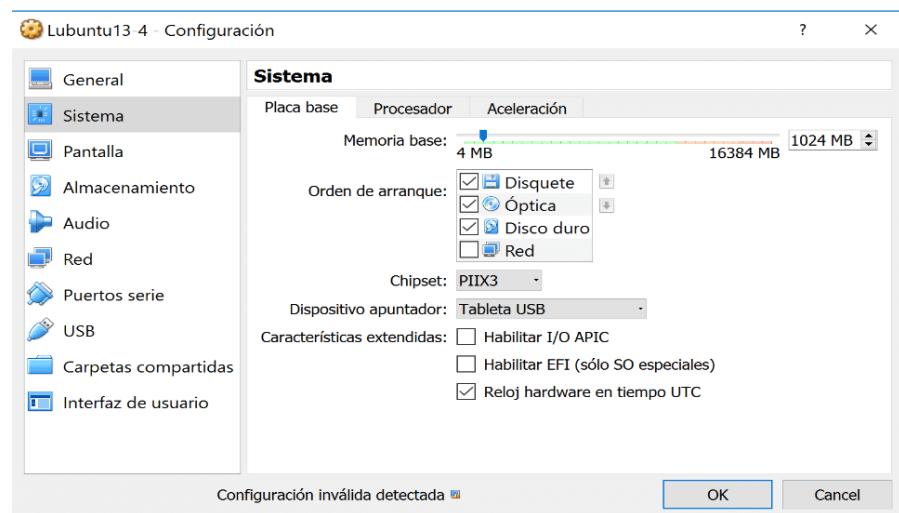


Figura 46. Configuración de la memoria RAM.

Fuente: elaboración propia.

A continuación, se selecciona en *Sistema* la opción *Placa Base* para fijar la cantidad de memoria de la máquina virtual (figura 46). En función de la memoria RAM que tenga el ordenador que se está usando, se puede aumentar o disminuir el valor. 512 Mb debería ser suficiente, pero se recomiendan 1024 Mb para que funcione con fluidez.

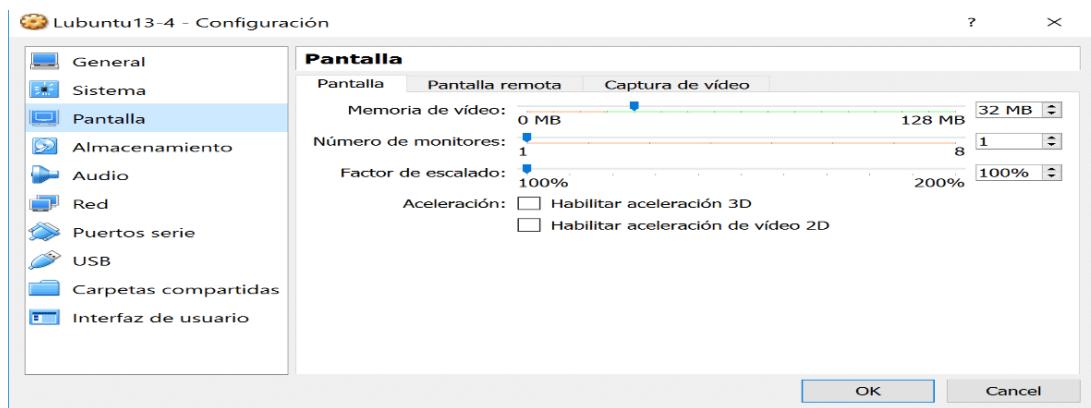


Figura 47. Configuración de la memoria para la tarjeta gráfica.

Fuente: elaboración propia.

Se selecciona la opción *Pantalla* y, en esta pestaña, se ajusta la cantidad de memoria para la tarjeta gráfica: 32 Mb debería ser suficiente (figura 47).

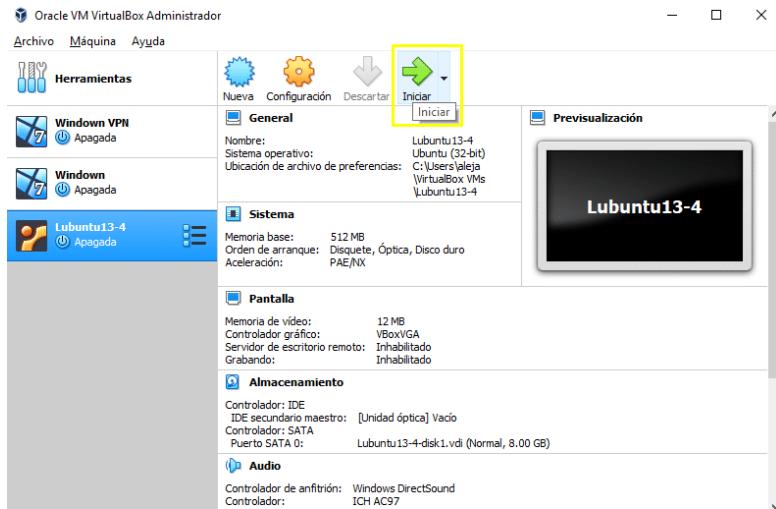


Figura 48. Inicio de la máquina virtual importada.

Fuente: elaboración propia.

A continuación, se pulsa en el botón OK. En la pantalla principal de Virtual Box, se selecciona la máquina virtual Lubuntu13-4 y se pulsa en el botón *Iniciar* (figura 48).

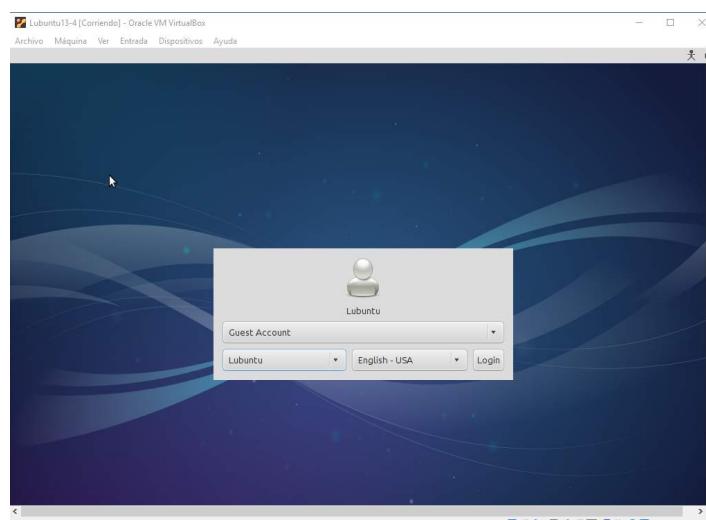


Figura 49. Login en la máquina virtual importada.

Fuente: elaboración propia.

A continuación, se ejecuta la máquina virtual Lubuntu13-4 y ya se podrá trabajar con ella. Se selecciona la cuenta “Guest Account” y se pulsa en *Login* (figura 49).

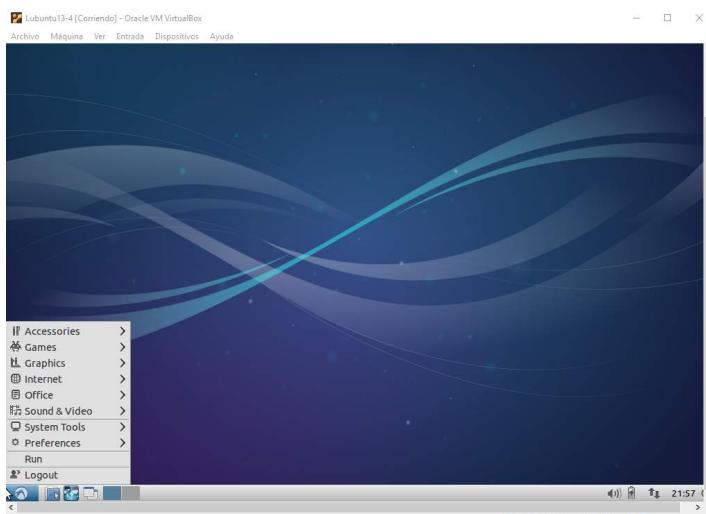


Figura 50. Escritorio de la máquina virtual importada.

Fuente: elaboración propia.

A continuación, aparece el escritorio del sistema operativo virtualizado (figura 50).

CONTINUAR

Algunas observaciones que se deben tener en cuenta:

- 1 Para que funcionen las máquinas virtuales, es necesario que el PC permita la virtualización hardware. Es una opción que suele encontrarse en el BIOS del equipo. Puede llamarse "Virtualización Hardware" o algo similar, en función del modelo del PC.
- 2 Para confirmar que el PC soportará la virtualización, se puede recurrir a la [siguiente página](#).
- 3 Si la importación da un error estilo ERROR_ZIP_CODE, muy posiblemente el archivo *.ova esté corrupto. Para solucionarlo, se debe descargar de nuevo la máquina virtual.
- 4 Una vez importada la máquina, si al intentar encenderla no funcionara, se debe revisar el punto 1.
- 5 También es importante que la máquina virtual esté preparada para el sistema operativo anfitrión, es decir, hay que saber si es de 64 o 32 bits.
- 6 Si al iniciar la máquina virtual aparece la pantalla con rayas negras o de colores, puede verificarse la solución del problema en el sitio web YouTube⁵.

⁵[Redes Plus. Instalar Ubuntu 20.04 en VirtualBox.](#) YouTube, agosto 2020.

CONTINUAR

Se acaba de describir cómo cargar una máquina virtual mediante la opción “*Importar un servicio virtualizado*”, puesto que la máquina se encontraba en un archivo de tipo *.ova. Sin embargo, las máquinas virtuales pueden distribuirse en otros formatos diferentes.

A continuación, se muestra **cómo importar una máquina virtual que se encuentra en un archivo de tipo *.vmdk**, el formato de virtualización de VMware⁶.

⁶[Descarga de máquinas en formatos VMware y VirtualBox.](#)

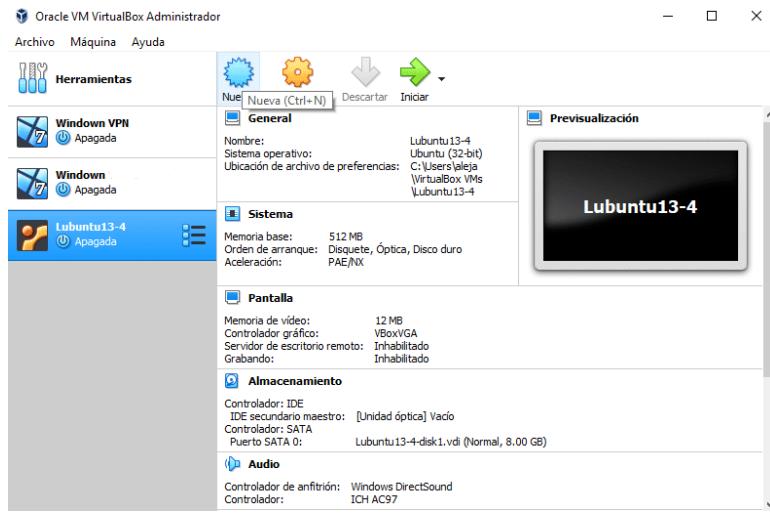


Figura 51. Importación de una máquina virtual tipo *.vmdk.

Fuente: elaboración propia.

Para ello, se pulsa en la opción *Nueva* (figura 51).

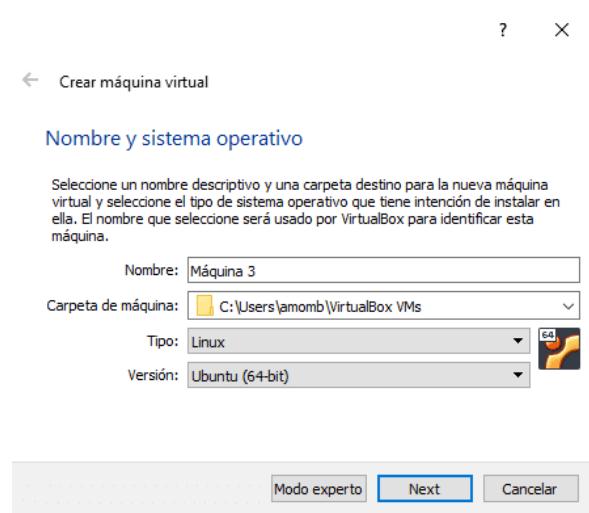


Figura52. Nombre de la máquina virtual.

Fuente: elaboración propia.

Aparece una nueva ventana en la que se debe configurar la máquina virtual. Se rellena la pestaña Nombre con uno cualquiera (por ejemplo, “Máquina 3”), en *Tipo* se selecciona “Linux” y en *Versión* se selecciona “Ubuntu (64-bit)”, tal como se ve en la figura 52.

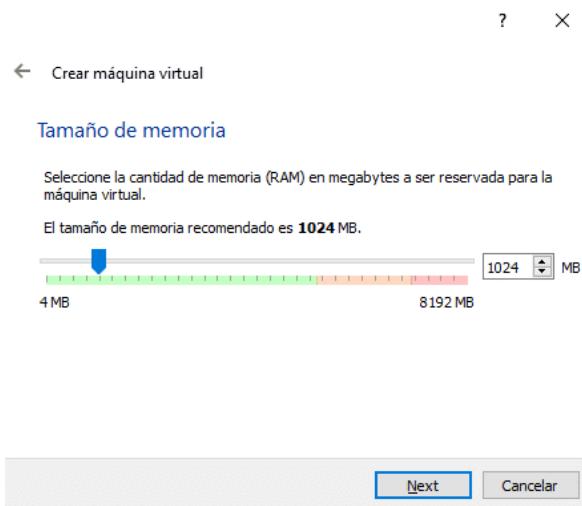


Figura 53. Configuración de la memoria.

Fuente: elaboración propia.

Se pulsa en *Next* y, en la ventana siguiente, se configura el tamaño de la memoria. 1024 Mb es un buen valor (figura 53).

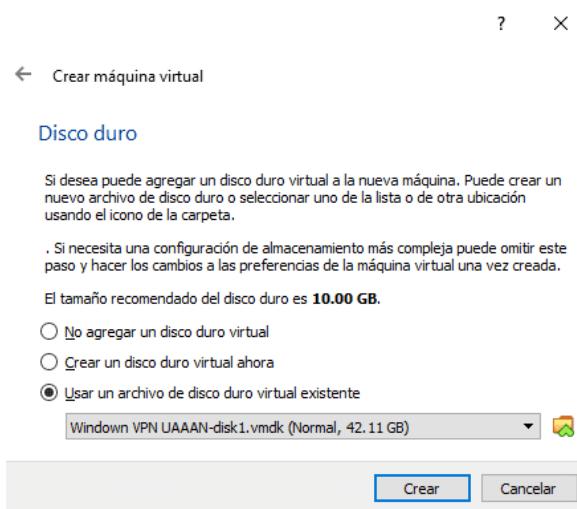


Figura 54. Seleccionar "Usar disco ya existente".

Fuente: elaboración propia.

A continuación, se pulsa en *Next* y en la ventana siguiente se selecciona la opción "Usar un archivo de disco duro virtual existente" y se pulsa sobre el icono en forma de carpeta (figura 54).

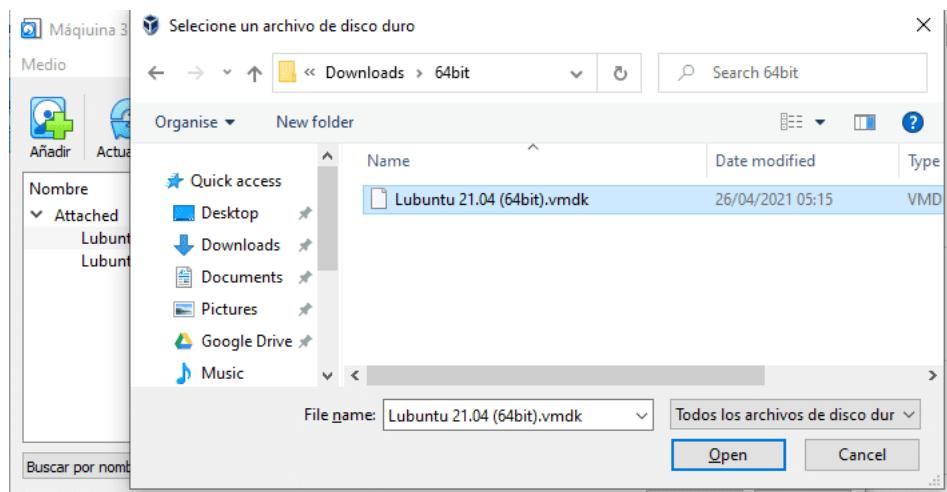


Figura 55. Búsqueda de la máquina virtual.

Fuente: elaboración propia.

Cuando se pulsa sobre el icono de la carpeta, se pulsa en Agregar y aparece un navegador de archivos, en el que se buscará la máquina que se quiere cargar (figura 55).

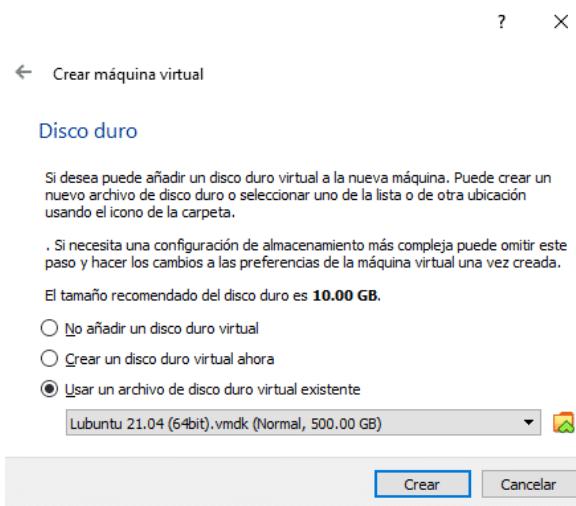


Figura 56. Máquina virtual seleccionada.

Fuente: elaboración propia.

Se pulsa en *Abrir* y se vuelve a la ventana anterior (figura 56).

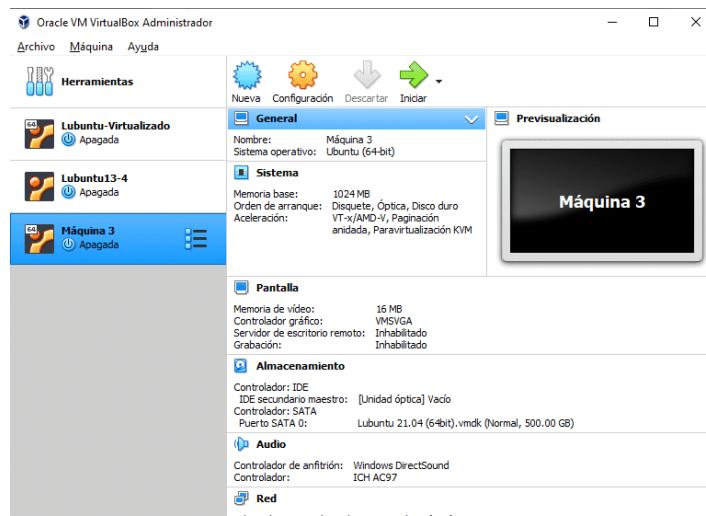


Figura 57. Máquina virtual importada.

Fuente: elaboración propia.

A continuación, se pulsa sobre *Crear* y aparecerá en el listado de máquinas virtuales (figura 57).

VI. La shell de comandos de Linux. Creación de scripts

6.1. La *shell* de comandos

Linux es un sistema operativo similar a Unix que se distribuye como software libre. Para ilustrar los contenidos de esta sección, se va a utilizar una distribución de Linux denominada **Lubuntu** que **se caracteriza por exigir pocos recursos hardware para su ejecución.**

Linux dispone de una aplicación denominada **intérprete de comandos** o **shell** de comandos que **permite al usuario interactuar con el sistema operativo mediante la ejecución de comandos o sentencias de texto**. Hay diferentes tipos de intérpretes que se diferencian en la sintaxis de los comandos y en la forma de interaccionar con el usuario.

Según el sistema Linux utilizado, habrá diferentes tipos de shell. **La más utilizada es Bash⁷ (Bourne Again Shell)**, que no solo permite ejecutar comandos puntuales, sino que **dispone de un sistema de sentencias donde se pueden componer scripts o automatismos de cierto grado de complejidad.**

⁷[GNU Bash.](#)

CONTINUAR

Para acceder al intérprete de comandos, se pulsa Ctrl+Alt+T o bien se pulsa en un ícono que aparece en la parte inferior izquierda, dentro de la barra de herramientas inferior del escritorio de Lubuntu (figura 58).

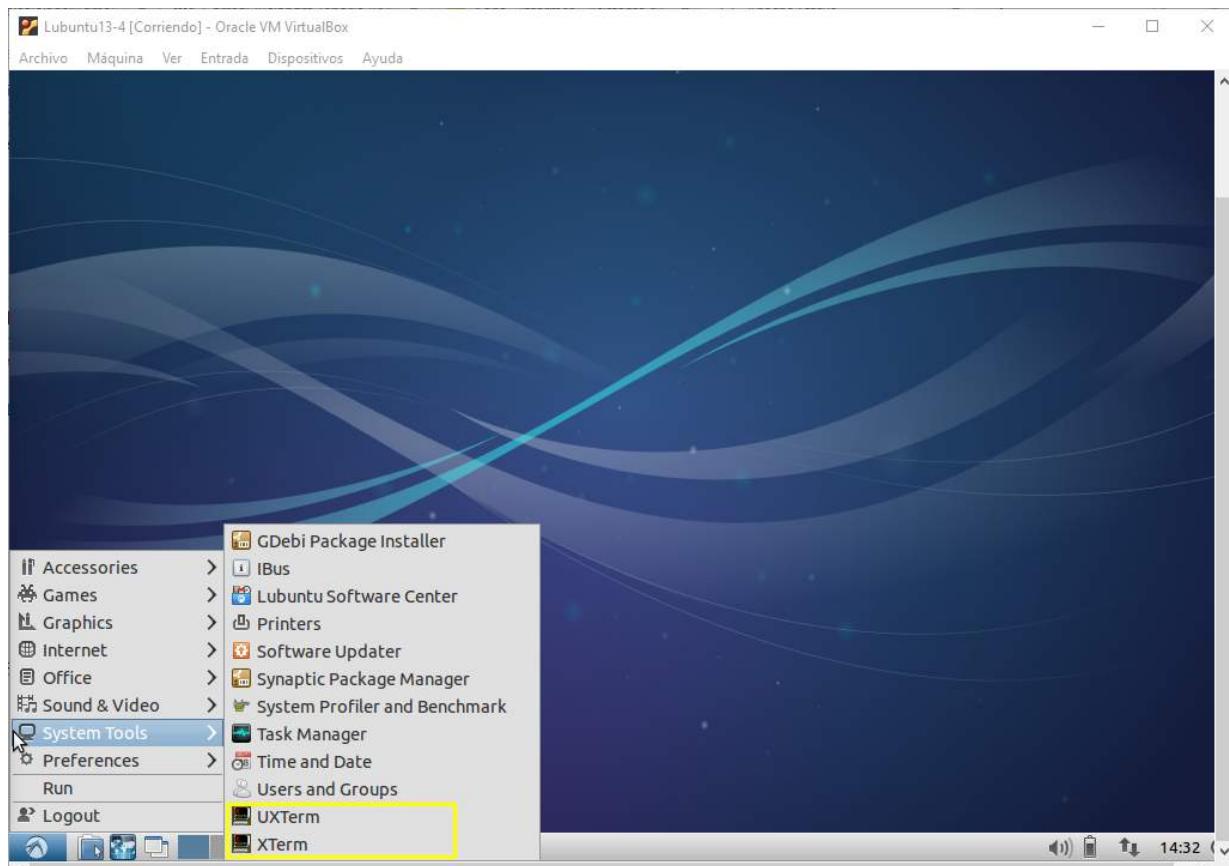


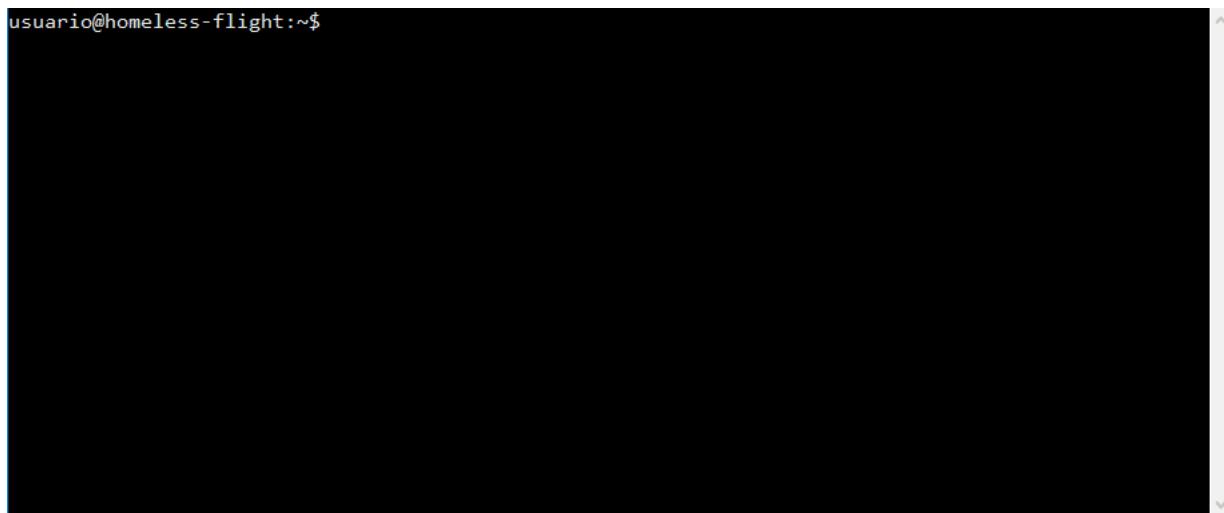
Figura 58. Acceso al intérprete de comandos de Linux.

Fuente: elaboración propia.

Algunas características generales son:

- El terminal muestra en pantalla un indicador de línea de órdenes, denominado “prompt”, para que el usuario introduzca una. El indicador finaliza con un carácter \$ en el caso de usuarios normales y con # en el caso del superusuario.
- Al comienzo de la línea de órdenes aparece el usuario en la forma usuario@máquina:directorio\$, donde:
 - “usuario” es el nombre del usuario logado.
 - “máquina” es el nombre asignado a la máquina.
 - “directorio” es la ruta, dentro del sistema de ficheros, en la que se encuentra el usuario en ese momento.
 - El carácter especial “~” se utiliza para referirse al “HOME” del usuario, que normalmente se corresponde con la ruta “/home/<usuario>”. Es el espacio de ficheros dentro del sistema global asignado a ese usuario (figura 59).
- Cuando se escribe un comando para que se ejecute, hay que pulsar la tecla **Enter**.
- Los comandos hay que teclearlos exactamente. En este sentido, las letras mayúsculas y minúsculas se consideran caracteres diferentes.

- Un amplio número de comandos de la *shell* está pensado para trabajar directamente con el **sistema de ficheros**⁸. Por tanto, es conveniente entender cómo se organiza este sistema y cómo la *shell* da información sobre él.
 - El directorio raíz es "/" y es el punto de partida hacia cualquier otro fichero o directorio del sistema.
 - La jerarquía de directorios⁹ se visualiza mediante rutas, donde se muestra el camino, de izquierda a derecha, que se ha seguido para llegar a un directorio. Por ejemplo, si hay un directorio "prueba1" en el HOME, el camino o "PATH" seguido será: "/home/usuario/prueba1".
- Los comandos pueden recibir parámetros que afectan a la manera de ejecutarse: comando parámetro1 parámetro2 ... parámetroN.
- Además de parámetros, los comandos pueden recibir modificadores propios, que van siempre precedidos de los símbolos "-" y "--". El primero se utiliza para indicar modificadores cuyo nombre es un único carácter, mientras que el segundo precede siempre a nombres completos de modificadores. Asimismo, es habitual que coexistan ambas versiones de un mismo modificador. Por ejemplo, un buen número de comandos ofrecen estos dos modificadores, que hacen lo mismo: **-v** y **--verbose** (hacen que la salida del comando contenga información adicional). Se podría utilizar uno u otro, indistintamente.



```
usuario@homeless-flight:~$
```

Figura 59. Intérprete de comandos Linux.
Fuente: elaboración propia.

⁸[Discussion on Linux Filesystems](#).

⁹[The Linux Directory Structure](#).

CONTINUAR

A continuación, se van a revisar algunos de los comandos que permiten navegar por el sistema de ficheros.

Cambio de directorio

```
usuario@homeless-flight:~$ cd ..
usuario@homeless-flight:/home$
```

Figura 60. Cambio de directorio.

Fuente: elaboración propia.

Para cambiar de directorio se usa el comando **cd**. Este comando recibe un único parámetro, que es el nombre del directorio al que se quiere ir: **cd directorio_destino**.

Asimismo, dentro de cada directorio existen dos directorios especiales: **“.”** y **“..”**. hace referencia al directorio actual, mientras que **“..”** se refiere al directorio padre del actual (figura 60).

Ubicación actual

```
usuario@homeless-flight:~$ cd ..  
usuario@homeless-flight:/home$ pwd  
/home  
usuario@homeless-flight:/home$
```

Figura 61. Uso del comando **pwd**.

Fuente: elaboración propia.

El comando **pwd** imprime la ruta del directorio actual (figura 61).

Listado del contenido de un directorio

```
usuario@homeless-flight:~$ cd ..  
usuario@homeless-flight:/home$ pwd  
/home  
usuario@homeless-flight:/home$ ls  
usuario  
usuario@homeless-flight:/home$ █
```

Figura 62. Uso del comando ls.

Fuente: elaboración propia.

El comando **ls** muestra los nombres de los archivos y subdirectorios contenidos en el directorio en el que se está (figura 62). Solo se obtienen los nombres de los archivos, sin ninguna otra información.

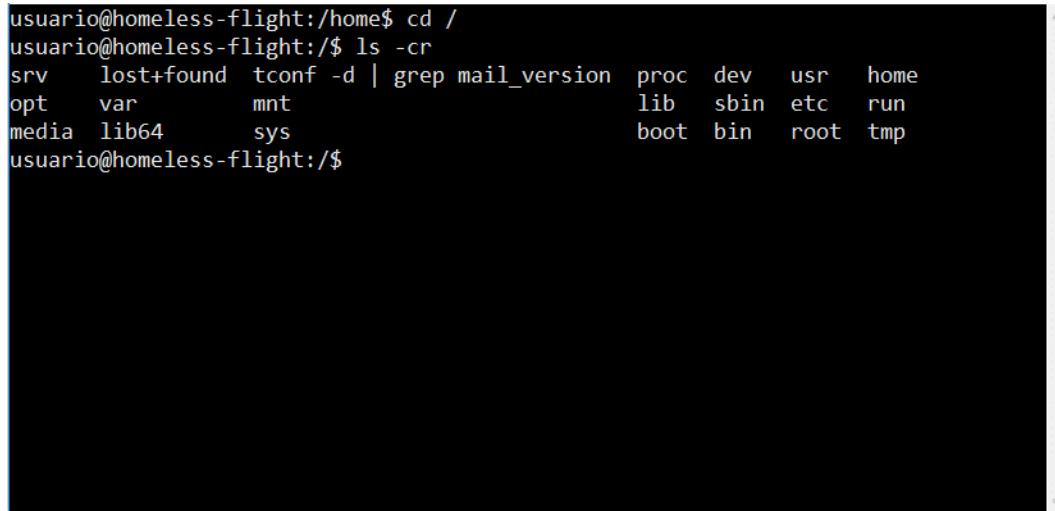
A terminal window with a black background and white text. The text shows the command 'ls -cr' being run in a directory. The output lists several system directories and files, such as 'lost+found', 'tconf', 'proc', 'dev', 'usr', 'home', 'lib', 'sbin', 'etc', 'run', 'boot', 'bin', 'root', and 'tmp'. The 'tconf' directory is shown with its contents: 'srv', 'opt', 'media', 'var', 'mnt', 'lib64', and 'sys'. The 'grep' command is used to filter the output, specifically looking for 'mail_version'.

Figura 63. Uso del comando **ls** con varias opciones.

Fuente: elaboración propia.

Admite diferentes modificadores que alterarán la información mostrada en pantalla por el comando:

- **ls -R** lista recursivamente los subdirectorios encontrados. Es decir, muestra los ficheros y directorios del actual, así como subdirectorios y su contenido.
- **ls -a** muestra todos los archivos, incluyendo algunos que están ocultos para el usuario —aquellos que comienzan por un punto —.
- **ls -c** muestra el contenido del directorio ordenado por día y hora de creación. – **ls -t** muestra el contenido del directorio ordenado por día y hora de modificación.
- **ls -r** muestra el contenido del directorio y lo ordena alfabéticamente en orden inverso.
- **ls subdir** muestra el contenido del subdirectorio "subdir".
- **ls -l filename** muestra toda la información sobre el archivo "filename".
- **ls -color** muestra el contenido del directorio coloreado: verde para los ejecutables, azul para las carpetas, fucsia para las imágenes, rojo para los comprimidos, etc.
- **ls -l** muestra toda la información de cada archivo, incluyendo permisos, tipo de archivo, tamaño y fecha de creación o del último cambio introducido, etc.

Las opciones anteriores pueden combinarse. Por ejemplo, **ls -cr** muestra el directorio ordenando inversamente por fechas (figura 63).

La *shell* de comandos admite el uso de caracteres denominados “comodín” (o *wildcards*), que permiten abarcar conjuntos de ficheros o directorios cuyo nombre encaje con la expresión que definen:

- El carácter “*” se utiliza para representar cualquier carácter un número indefinido de veces.
- El carácter “?” se utiliza para representar cualquier carácter una única vez.

Si hay dos ficheros, “bar.txt” y “baz.txt”, se podrían listar ambos (sin considerar el resto de los ficheros del directorio) mediante el comando **ls ba?.txt**.

Asimismo, si se quisieran listar todos los ficheros con extensión .gif, podría hacerse mediante el comando **ls *.gif**.

Existe otro comando, denominado **dir**, que tiene la misma función que **ls**, pero sin mostrar tanta información.

Creación de ficheros

El comando **touch** actualiza los registros de fecha y hora con la fecha y hora actual de los ficheros indicados como argumento. Si el fichero no existe, el comando **touch** lo crea. Su uso más frecuente es para crear archivos.

La sintaxis del comando **touch** sigue la forma: **touch fichero**.

Además, **touch ejemplo.txt** crea el archivo "ejemplo.txt" en el directorio actual si este no existiera.

Creación de subdirectorios

```
usuario@homeless-flight:~$ ls
usuario@homeless-flight:~$ mkdir prueba
usuario@homeless-flight:~$ ls
prueba
usuario@homeless-flight:~$
```

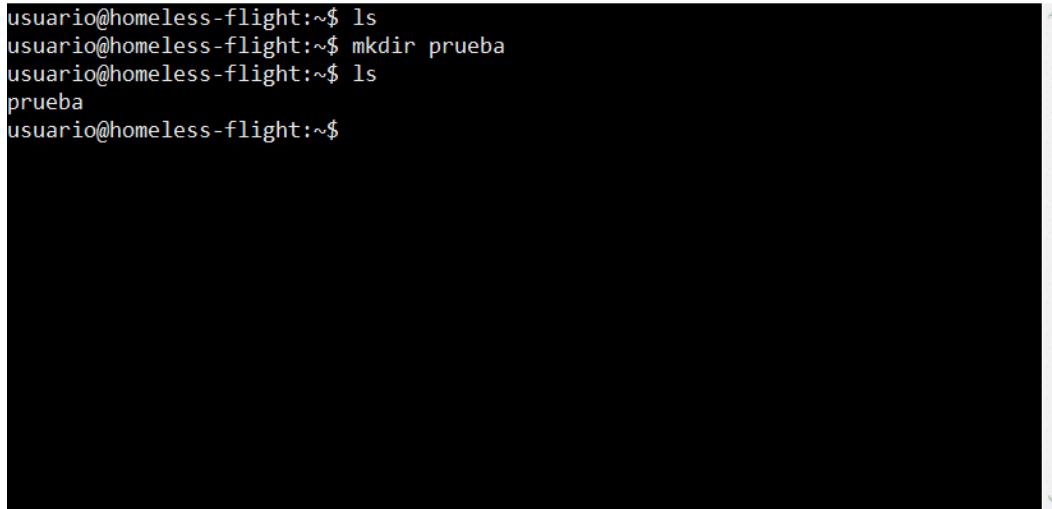


Figura 64. Creación de un subdirectorio.

Fuente: elaboración propia.

El comando **mkdir** permite crear un nuevo subdirectorio. La sintaxis es **mkdir subdir1** donde **subdir1** es el nombre del directorio que se va a crear (figura 64).

Borrado de subdirectorios

```
usuario@homeless-flight:~$ ls
prueba
usuario@homeless-flight:~$ rm -R prueba/
usuario@homeless-flight:~$ ls
usuario@homeless-flight:~$
```

Figura 65. Borrado de un subdirectorio.

Fuente: elaboración propia.

El comando **rmmdir** borra uno o más directorios del sistema siempre que estos subdirectorios estén vacíos. La sintaxis es **rmmdir subdir1** donde **subdir1** es el nombre del directorio que se va a eliminar (figura 65).

Copia de archivos

```
usuario@homeless-flight:~$ ls  
file.txt  
usuario@homeless-flight:~$ cp file.txt file2.txt  
usuario@homeless-flight:~$ ls  
file2.txt  file.txt  
usuario@homeless-flight:~$
```

Figura 66. Copia de un archivo.

Fuente: elaboración propia.

La sintaxis del comando es **cp file1 file2** (figura 66), donde file1 y file2 indican el nombre de un archivo o su ruta. Al ejecutarlo, se crea una copia de “file1” que se denomina “file2”. Si “file2” no existía, lo crea con los mismos atributos de “file1” y, en caso de existir, su contenido se sustituye por el de “file1”. El archivo “file2” estará en el mismo directorio que “file1”.

```
usuario@homeless-flight:~$ mkdir prueba
usuario@homeless-flight:~$ cp file.txt file2.txt prueba/
usuario@homeless-flight:~$ ls prueba/
file2.txt  file.txt
usuario@homeless-flight:~$ █
```

Figura 67. Copia de un archivo en un directorio determinado.

Fuente: elaboración propia.

También se puede usar como **cp file1 file2 namedir** que hace copias de “file1” y “file2” en el directorio “namedir” (figura 67).

Traslado y cambio de nombre de archivos

```
usuario@homeless-flight:~$ ls
file2.txt file.txt prueba
usuario@homeless-flight:~$ mv file.txt file3.txt
usuario@homeless-flight:~$ ls
file2.txt file3.txt prueba
usuario@homeless-flight:~$
```

Figura 68. Cambio del nombre de un archivo.

Fuente: elaboración propia.

El comando **mv** permite “mover” un fichero o directorio a otro lugar del sistema de ficheros, de manera que, una vez ejecutado, dejará de existir en su ubicación original. Recibe dos parámetros: el fichero origen y el fichero destino.

La sintaxis es **mv fichero_origen fichero_destino** (figura 68).

Si los nombres que aparecen son de directorios, entonces el comando **mv namedir1 namedir2** cambia el nombre del subdirectorio “namedir1” por “namedir2”.

El comando **mv file1 file2 namedir** traslada uno o más archivos (“file1”, “file2”, etc.) al directorio “namedir” y conserva el nombre.

Borrado de archivos

```
usuario@homeless-flight:~$ ls
file2.txt file.txt prueba
usuario@homeless-flight:~$ mv file.txt file3.txt
usuario@homeless-flight:~$ ls
file2.txt file3.txt prueba
usuario@homeless-flight:~$ rm file3.txt
usuario@homeless-flight:~$ ls
file2.txt prueba
usuario@homeless-flight:~$
```

Figura 69. Borrado de un archivo.

Fuente: elaboración propia.

El comando **rm** elimina uno o más archivos de un directorio en el cual se tenga permiso de escritura. La sintaxis es **rm file1 file2** (figura 69). En el comando, se pueden utilizar los caracteres comodín * y ?.

```
usuario@homeless-flight:~$ ls
file2.txt prueba
usuario@homeless-flight:~$ rm -i file2.txt
rm: remove regular file 'file2.txt'? y
usuario@homeless-flight:~$ ls
prueba
usuario@homeless-flight:~$
```

Figura 70. Borrado de un archivo con opción **-i**.

Fuente: elaboración propia.

Si se usa la opción **-i**, se pide confirmación para borrar cada archivo de la lista especificada: **rm -i file1 file2** (figura 70).

Enlaces (links) a archivos

```
usuario@homeless-flight:~$ ls
prueba prueba.txt
usuario@homeless-flight:~$ ln prueba.txt prueba2.txt
usuario@homeless-flight:~$ ls
prueba prueba2.txt prueba.txt
usuario@homeless-flight:~$ ls -lt
total 12
-rw-rw-r-- 2 usuario usuario    6 Feb 18 18:11 prueba2.txt
-rw-rw-r-- 2 usuario usuario    6 Feb 18 18:11 prueba.txt
drwxrwxr-x 2 usuario usuario 4096 Feb 18 17:57 prueba
usuario@homeless-flight:~$
```

Figura 71. Enlace a un archivo.

Fuente: elaboración propia.

Un mismo archivo puede estar repetido con más de un nombre. También es posible tener un mismo archivo con varios nombres distintos y acceder a él desde más de un directorio. Esto último se denomina enlaces múltiples a un archivo y se crean mediante el comando **ln**, cuya sintaxis es: **ln file1 file2**.

De esta forma, el archivo “file1” tiene dos nombres: file1 y file2. Además, este comando advierte previamente si el nombre file2 está ocupado y en este caso no se ejecuta.

En el ejemplo se crea un enlace al archivo “prueba.txt” (figura 71).

Los archivos enlazados a otro se borran como los archivos normales, de manera que, si se borra el archivo original, su contenido permanece en los archivos enlazados.

Características de un archivo

```
usuario@homeless-flight:~$  
usuario@homeless-flight:~$ file prueba.txt  
prueba.txt: ASCII text  
usuario@homeless-flight:~$ █
```

Figura 72. Características de un archivo.

Fuente: elaboración propia.

Mediante el comando **file** se puede obtener información acerca de un archivo: **file nombre_fichero** (figura 72).

Cambio de modo en los archivos

Llegado este punto, conviene hacer un alto en el camino para comentar una de las principales características de los sistemas Linux: están concebidos para que los utilicen diferentes usuarios de manera simultánea. Es decir, son multiusuario.

Por tanto, se hace imperativo definir un sistema de permisos que limite la propiedad y el uso que se hace de los ficheros que pertenecen a cada usuario, de manera que cada uno pueda decidir con quién comparte cada fichero dentro del propio sistema de ficheros.

Se definen por tanto tres tipos de permiso¹⁰:

- **Lectura**, el usuario puede leer el contenido del fichero.
- **Escritura**, el usuario puede modificar el contenido del fichero.
- **Ejecución**, si el fichero es un script (contiene sentencias de comandos) o un programa ejecutable, el usuario puede ejecutarlo.

¹⁰[Linux Users and Groups](#).

```
usuariov@usuario-virtualbox:~$ ls -l
total 40
-rw-rw-r-- 1 usuariov usuariov 24 may 22 20:10 bar.txt
drwxr-xr-x 2 usuariov usuariov 4096 may 21 10:50 Descargas
drwxrwxr-x 2 usuariov usuariov 4096 may 22 20:08 Desktop
drwxr-xr-x 2 usuariov usuariov 4096 may 21 10:50 Documentos
-rw-rw-r-- 1 usuariov usuariov 11 may 22 20:10 foo.txt
drwxr-xr-x 2 usuariov usuariov 4096 may 21 10:50 Imágenes
drwxr-xr-x 2 usuariov usuariov 4096 may 21 10:50 Música
drwxr-xr-x 2 usuariov usuariov 4096 may 21 10:50 Plantillas
drwxr-xr-x 2 usuariov usuariov 4096 may 21 10:50 Público
drwxr-xr-x 2 usuariov usuariov 4096 may 21 10:50 Videos
usuariov@usuario-virtualbox:~$
```

Figura 73. Listado extenso.

Fuente: elaboración propia.

De la misma forma, se definen tres ámbitos de aplicación de esos permisos:

- **Owner**, el permiso se aplica al propietario del fichero.
- **Group**, el permiso se aplica a un grupo de usuarios (los usuarios pueden pertenecer a varios grupos).
- **Global**, el permiso se aplica a todos los usuarios del sistema.

Los permisos de cada archivo se pueden ver con el comando **ls -l**.

La salida de este comando ofrece una lista de ficheros con la siguiente información (figura 73):

- La primera columna muestra el tipo de fichero y sus permisos.
- La segunda muestra el número de enlaces a ese fichero.
- La tercera indica qué usuario es el propietario del fichero.
- La cuarta indica el grupo al que pertenece el fichero.
- La quinta muestra el tamaño en bytes del fichero. Los directorios, según qué sistema de ficheros se use, suelen ocupar un bloque de datos completo¹¹ (4096 bytes).
- Las columnas sexta a octava muestran la fecha y hora de última modificación.
- La novena muestra el nombre del fichero.

¹¹[Why do directories have a size of 4096 bytes.](#)

Desgranando la columna de permisos:

- El primer carácter indica el tipo de fichero: “d” se trata de un directorio, “l” un enlace simbólico y “-” un fichero ordinario.
- Los siguientes caracteres definen todos los permisos, agrupados en tres conjuntos de tres caracteres cada uno. El primer grupo define los permisos del usuario propietario del fichero. El segundo los del grupo y el tercero los permisos globales.

Cada grupo de permisos se define mediante los caracteres “rwx”, donde r indica permiso de lectura, w de escritura y x de ejecución. Si en una de esas posiciones aparece el carácter “-”, quiere decir que esa entidad (usuario, grupo o global) no tiene el permiso que se define en esa posición.

Ejemplo

Tomando el fichero “bar.txt” de la figura 1.71, -rw-rw-r--:

- Se trata de un fichero ordinario (el primer carácter es “-”).- El propietario, **usuariov**, puede leer y modificar su contenido, pero no puede ejecutarlo (**rw-**).
- El grupo al que pertenece, también llamado **usuariov**, puede leer y modificar su contenido, pero no puede ejecutarlo (**rw-**).
- El resto de los usuarios solo pueden leer su contenido (**r--**).

```

usuario@homeless-flight:~$ ls -lt
total 12
--w--w--- 2 usuario usuario      6 Feb 18 18:11 prueba2.txt
--w--w--- 2 usuario usuario      6 Feb 18 18:11 prueba.txt
drwxrwxr-x 2 usuario usuario 4096 Feb 18 17:57 prueba
usuario@homeless-flight:~$ chmod +x prueba.txt
usuario@homeless-flight:~$ ls -lt
total 12
--wx-wx--x 2 usuario usuario      6 Feb 18 18:11 prueba2.txt
--wx-wx--x 2 usuario usuario      6 Feb 18 18:11 prueba.txt
drwxrwxr-x 2 usuario usuario 4096 Feb 18 17:57 prueba
usuario@homeless-flight:~$
```

Figura 74. Cambio de propiedades de un archivo.

Fuente: elaboración propia.

Para cambiar los permisos de un archivo se emplea el comando **chmod** [quien] **oper**, **permiso**, **files**, donde:

- **quien** indica a quién afecta el permiso que se desea cambiar. Es una combinación de las letras **u** para el usuario, **g** para el grupo del usuario, **o** para los otros usuarios y **a** para todos los anteriores. Si no se indica nada, se supone **a**.
- **oper** indica si el permiso se da usando un **+** o se quita usando un **-**.
- **permiso** indica el permiso que se quiere dar o quitar. Es una combinación de las letras: **r** (leer), **w** (escribir), **x** (ejecutar). Estos permisos son diferentes cuando se aplican a directorios: **r** da la posibilidad de ver el contenido del directorio con el comando **ls**; el permiso **w** da la posibilidad de crear y borrar archivos en ese directorio; el permiso **x** autoriza a buscar y utilizar un archivo concreto.
- **files** son los nombres de los archivos o directorios cuyos modos de acceso se quieren cambiar.

En el ejemplo, se cambian los permisos al archivo "prueba.txt" (figura 74).

CONTINUAR

Cambio de propietario a un conjunto de archivos

El comando **chown** se emplea para cambiar de propietario a un determinado conjunto de archivos: **chown newowner file1 file2...**

La lista completa de usuarios del sistema se puede ver en el fichero "/etc/passwd".

Cambio de grupo de un archivo

El comando **chgrp** se emplea para cambiar el grupo al que pertenece un archivo: chgrp **newgroup file1 file2...**

Los grupos de usuarios están almacenados en el archivo "/etc/group".

Visualización sin formato de archivos

El comando **cat filename** permite visualizar el contenido de uno o más archivos de forma no formateada y también permite copiar uno o más archivos como apéndice de otro ya existente:

- **cat filename** saca por pantalla el contenido del archivo "filename".
- **cat file1 file2...** saca por pantalla, secuencialmente y según el orden especificado, el contenido de los archivos indicados.
- **cat file1 file2 >file3**, el contenido de los archivos "file1" y "file2" se almacena en "file3".
- **cat file1 file2 >>file3**, el contenido de "file1" y "file2" se añade al final de "file3".
- **cat >file1** acepta lo que se introduce por el teclado y lo almacena en "file1" (se crea "file1"). Para terminar, se emplea <ctrl>d.

Alternativamente, se puede usar el editor **vi**.¹² El editor puede encontrarse en dos estados o modos:

COMANDOS	EDICIÓN
En el modo de comandos , vi está esperando alguna orden en forma de pulsación de teclas o combinaciones de teclas concretas (por tanto, interpreta lo que se escribe como órdenes).	

En el modo de **edición**, **vi** está esperando que se escriba el texto del fichero (por tanto, interpreta lo escrito como texto).

¹² [The Vi Editor.](#)



Cuando se entra en **vi**, está en modo de comandos. Para pasar al modo de edición, se puede pulsar **i** (insertar), **a** (añadir), **o** (añadir una línea). Para pasar al modo de comandos, se puede pulsar **Escape** o **Suprimir**.

Comandos de vi

—

Algunos comandos de vi son:

- **i** insertar antes del cursor
- **a** añadir detrás del cursor
- **o** añadir una línea en blanco
- **x** borrar un carácter
- **j** borrar el final de línea (une dos líneas)
- **dd** borra la línea completa
- **u** deshacer la última edición
- **:q** salir
- **:q!** salir sin guardar
- **:w** guardar
- **:wq** guardar y salir
- **:set nu** muestra números de línea
- **:set nonu** oculta números de línea

CONTINUAR

Visualización de archivos con formato

El comando **pr file** imprime por consola el contenido de los archivos de una manera formateada, por columnas, controlando el tamaño de página y poniendo cabeceras al comienzo:

- **pr file** produce una salida estándar de 66 líneas por página, con un encabezamiento de 5 líneas: 2 en blanco, 1 de identificación y otras 2 líneas en blanco.
- **pr -ln file** produce una salida de n líneas por página.
- **pr -F file** hace una pausa para presentar la página, hasta que se pulsa Return para continuar.
- **pr -t file** suprime las 5 líneas del encabezamiento y las del final de página.
- **pr -wn file** ajusta la anchura de la línea a n posiciones.
- **pr -d file** lista el archivo con espaciado doble.
- **pr -h `caracteres` file**, el argumento o cadena de caracteres `caracteres` se convertirán en la cabecera del listado.
- **pr +n file** imprime el archivo a partir de la página n.

Anotación:

Se pueden combinar varias opciones en un mismo comando, como por ejemplo **pr -dt file** o cambiar la salida con **pr file1 > file2**, que crea un archivo nuevo llamado file2 que es idéntico a "file1", pero con formato por páginas y columnas.

CONTINUAR

Visualización completa de archivos

CAT

MORE

LESS

Muestra el contenido completo de uno o más ficheros: **cat fichero1 fichero2...**

CAT	MORE	LESS
-----	------	------

Muestra el contenido del fichero de forma paginada, de manera que se puede navegar (siempre hacia delante) a lo largo de ficheros grandes, página a página: **more fichero**.

Al final de cada página se mostrará el texto **--More(x%)--**, donde x será el porcentaje del documento que ya se ha recorrido. Para pasar a la siguiente página, hay que pulsar la barra espaciadora. Y para salir, basta con pulsar **Ctrl+d o q**.

CAT	MORE	LESS
-----	------	------

es similar a **more**, pero permite navegar hacia delante y hacia atrás en el documento y hacerlo línea a línea: **less fichero**. Para navegar, basta con pulsar las teclas **Flecha arriba** y **Flecha abajo**.

Búsqueda en archivos

El comando **grep 'conjuntocaracteres' file1 file2 file3** busca una palabra, clave o frase en un conjunto de ficheros y muestra las líneas que contienen el texto buscado. Si el conjunto de caracteres está compuesto por dos o más palabras separadas por un espacio, se debe escribir entre apóstrofes, ', para evitar que la consola interprete esas palabras como otros parámetros del comando **grep**.

Por defecto, **grep** muestra las líneas que contienen el texto que debe buscar, pero también ofrece modificadores que permiten obtener otro tipo de información:

- **grep -c "texto a buscar" fichero1 ...ficheroN**, para cada fichero, muestra un número indicando el número de líneas que contienen ese texto.
- **grep -i "Texto a Buscar" fichero1 ...ficheroN** muestra las líneas que contienen el texto buscado, sin tener en cuenta mayúsculas y minúsculas. En este ejemplo, se considerarán textos válidos "texto a buscar" y "TeXTO a BuScar".
- **grep -l "Texto a Buscar" fichero1 ...ficheroN** muestra solo los nombres de los ficheros que contienen alguna ocurrencia del texto de búsqueda.
- **grep -n "Texto a Buscar" fichero1 ...ficheroN**, además de imprimir las líneas que coinciden con la búsqueda, se imprime su posición en el fichero.
- **grep -s "Texto a Buscar" fichero1 ...ficheroN** evita que se impriman en pantalla los mensajes de error que se puedan producir al no poder procesar un fichero (porque esté corrupto o porque no se tengan suficientes permisos, por ejemplo).
- **grep -v "Texto a Buscar" fichero1 ...ficheroN** es búsqueda inversa, es decir, se imprimen todas las líneas que NO contengan el texto buscado.

- **grep -e “/expresión regular/” fichero1 ...ficheroN**, en lugar de buscar ocurrencias completas del texto, busca aquellas líneas que cumplan con la expresión regular definida.¹³

¹³[Regular Expressions](#).

[Regular Expressions Tutorial](#).

[Validador de expresiones regulares](#).

CONTINUAR

Impresión en papel

El comando **lpr nombre_archivo** imprime por defecto el archivo indicado. Si se usa sin argumentos, imprime el texto que se introduzca a continuación en la impresora.

Compresión y agrupación de archivos

Comando gzip

Comprime los ficheros que recibe como parámetros, eliminando los originales.

gzip fichero1 creará en el mismo directorio la versión comprimida de “fichero1” (“fichero1.gz”) y eliminará “fichero1”.

Comando gunzip

Realiza la operación inversa sobre los archivos comprimidos que recibe como parámetro, restaurando los originales.

Comando tar

Empaqua un conjunto de ficheros o directorios en un único fichero con extensión .tar. Este comando presenta varios modificadores, que permiten construir un paquete o extraer su contenido, así como mostrar información adicional:

- **-c** indica que se debe crear un nuevo paquete .tar.
- **-x** indica que se debe extraer el contenido del fichero .tar recibido como parámetro.
- **-v** indica que se debe mostrar información extra sobre los ficheros que se están procesando.
- **-f** indica que el parámetro a continuación será el nombre del fichero.tar que se va a crear.

Algunos ejemplos

- **tar -cvf nuevo_fichero.tar fichero1 directorio2 fichero3** crea "nuevo_fichero.tar", que contendrá "fichero1", todo el "directorio2" (y sus subdirectorios) y "fichero3".
- **tar -xvf fichero.tar** extraerá el contenido de "fichero.tar" y lo dejará en el directorio en el que esté.

Anotación:

Los comandos **tar** y **gzip** se pueden combinar con la opción **-z** del comando **tar**. Por ejemplo: **tar -czvf nuevo_fichero.tar.gz fichero1 directorio2 fichero3** primero creará el paquete "nuevo_fichero.tar" y después lo comprimirá con gzip, dejando únicamente "nuevo_fichero.tar.gz".

Redirecciones¹⁴

Los comandos tienen una entrada estándar (denominada **stdin** y referida comúnmente con el número 0) y dos salidas estándar (**stdout** o número 1 para la salida normal y **stderr** o número 2 para la salida con errores). De esta manera, cada comando recibe datos desde **stdin**, los procesa y muestra el resultado de su ejecución, bien en **stdout** o **stderr** (o una combinación de ambos). Por defecto, y para la mayoría de los comandos, **stdin** recoge los valores introducidos desde teclado, a menos que se especifique un fichero para la entrada de datos (como en el comando **cat fichero**).

Asimismo, las salidas **stdout** y **stderr** muestran su contenido por defecto en la ventana del terminal (por ejemplo, **cat fichero** muestra el contenido del fichero en el terminal).

Sin embargo, se puede redirigir la salida de un comando usando los operadores:

- (>) redirige la salida estándar hacia el archivo indicado y, en caso de no existir, se crea.
- (<) redirige la entrada estándar desde el contenido de un determinado archivo.
- (>>) redirige la salida estándar hacia otro archivo, pero añadiendo dicha salida al final de ese archivo, sin sobrescribir el contenido original.

Ejemplo

date > fichero. Cada vez que se ejecuta este comando, se añade la fecha actual al final de “fichero”.

cat fichero1 fichero2 > fichero3 combina de manera secuencial “fichero1” y “fichero2” y guarda el resultado en “fichero3”.

cat fichero1 >> fichero2 realiza la misma operación que el anterior, pero añadiendo el contenido de “fichero2” al final de “fichero1”.

CONTINUAR

Tuberías

Una tubería (|) permite comunicar la salida estándar de un comando con la entrada estándar de otro. Por ejemplo, **ls | mail juan** envía a juan una lista de los archivos del sistema. Con el operador de tubería se pueden empalmar tantos comandos como se quiera.

Bifurcación

Permite redirigir la salida de un comando a un determinado archivo y que también se bifurque hacia el terminal. Para ello se usa el operador **tee**, como por ejemplo **ls | tee fichero**, que muestra la salida por el terminal y además la envía al archivo “fichero”. En este ejemplo, si se quiere que la salida se añada al final de “fichero”,

se usaría la opción **-a (append)**: **ls | tee -a file**.

Redirección de los errores

Los mensajes de error se dirigen a la salida número 2, que por defecto es el terminal. En algunos casos puede interesar redirigir estos mensajes a otra salida, por ejemplo, cuando se ejecuta un programa en segundo plano (es decir, al margen del terminal). Véanse a continuación los siguientes ejemplos:

gcc prueba.c 2>errores —

Redirige la salida 2 hacia el archivo "errores". De esta manera, el fichero "errores" contendrá los errores emitidos por el comando **gcc**, mientras que **stdout** se seguirá mostrando en la ventana del terminal.

program resultados.r 2> & 1 —

Redirige la salida estándar de errores al mismo archivo que la salida estándar.

CONTINUAR

Ejecución de un programa

Existen varios comandos para gestionar la ejecución de un programa.

Carácter & —

El carácter **&** al final de un comando permite ejecutarlo en segundo plano. Esto significa que, tras pulsar Enter, el comando comienza su ejecución; pero, además, se recupera inmediatamente el control del terminal, donde se pueden seguir ejecutando otros comandos mientras se ejecuta el anterior. Por ejemplo: program **<datos.d >resultados.r &**.

Comando kill

—

Para detener la ejecución de un proceso se puede utilizar el comando **kill <pid_del_proceso>**. El **pid** (o *process identifier*) es un identificador único que el sistema asigna a cada proceso.

Comando ps

—

Para conocer el **pid** de un proceso en ejecución basta con ejecutar el comando **ps**. Por ejemplo, **ps -ef** mostrará una lista completa de procesos, incluyendo detalles como pid de cada proceso, usuario, comando, etc.

Comando nohup program

—

Cuando se sale del sistema, si hay algún proceso ejecutándose en segundo plano, este se para salvo que se use el comando **nohup program**. En este caso, si no se utilizan redirecciones, todas las salidas del programa se dirigen a un archivo llamado "nohup.out".

Comando nice

—

El comando **nice** permite realizar ejecuciones con baja prioridad¹⁵ de la forma:

- **nice program &**
- **nice nohup program &**

Para darle al programa la prioridad mínima, habría que invocarlo como **nice -19 program &**, donde el -19 indica la mínima prioridad.

¹⁵"[A Guide to Priority and nice values in linux](#)".

Comando time

—

El comando **time**, cuando precede a cualquier otro comando, suministra información acerca del tiempo total empleado en la ejecución, del tiempo de CPU utilizado por el programa del usuario y del tiempo de CPU consumido en utilizar recursos del sistema. Por ejemplo, **time gcc prueba.c** ofrece información sobre el tiempo de compilación y montaje del programa prueba.c.

Comando top

—

El comando **top** muestra una lista de los procesos que se están ejecutando. Sus principales opciones son **u**, que muestra los procesos que pertenecen a un determinado usuario), **k**, que elimina un proceso, y **h**, que muestra la ayuda del programa.

Otros comandos

En la tabla 1.1 se muestran **otros comandos interesantes** del intérprete de comandos de Linux.

Comando	Significado
date	Muestra el día y la hora.
cal	Muestra el calendario. Tiene diversas opciones. Por ejemplo, cal 1945 mostraría el calendario del año 1945.
who	Indica qué usuarios tiene el ordenador en ese momento, en qué terminal están y desde qué hora.
whoami	Indica cuál es el terminal y la sesión en que se está trabajando.
uptime	Muestra el tiempo que lleva encendido el ordenador.
hostname	Muestra el nombre de la máquina.
bc	Abre una calculadora de texto. Para salir se introduce quit .
lshw	Muestra los diferentes dispositivos PCI conectados al sistema.
umask	Muestra los permisos con los que el usuario creará sus directorios por defecto.
uname -r	Muestra la versión del kernel.
lsusb	Muestra información de los dispositivos conectados en los puertos USB.
xkill	Permite cerrar un programa bloqueado.
groups	Muestra los grupos del sistema a los que pertenece un usuario.
clear	Limpia la consola.
head	Indicando un número n y un nombre de fichero, muestra las n primeras líneas del fichero indicado.
Tabla 1. Comandos de Linux. Fuente: elaboración propia.	



CONTINUAR

6.2. Creación de scripts

Un guion o script es un archivo de texto que contiene comandos de la *shell* que se interpretarán cuando se ejecute el script. Se caracteriza por:

- Pueden tener cualquier nombre.
- El archivo debe tener permiso de ejecución.
- Para ejecutarlo se debe indicar el nombre del intérprete y el nombre del script.
- Las instrucciones del script se procesan por orden, una por línea, de manera que los saltos de línea se interpretan como un “INTRO”.
- Las órdenes podrían aparecer en una misma línea separadas por punto y coma.

Los guiones **pueden tener una línea inicial que indica el tipo de intérprete que se quiere utilizar para ejecutar los comandos**. Por ejemplo, `#!/bin/bash` indica que debe usarse el intérprete Bourne-Again Shell.

Asimismo, un script **puede contener comentarios** que se marcan con el carácter `#` al inicio del texto del comentario.

Los principales elementos de un script son:

CONTINUAR

Variables y parámetros

Para definir una variable se usa la estructura `variable=valor`, donde `variable` es el nombre de esta. Hay que tener en cuenta lo siguiente:

1

No puede haber un espacio entre el nombre de la variable, el signo = y el valor.

2

Si se desea que el valor contenga espacios, es necesario utilizar comillas.

3

Para recuperar el valor de una variable hay que anteponerle a su nombre el carácter \$. Por ejemplo, para visualizar el valor de una variable `echo $variable`.

El comando `echo` permite imprimir cualquier carácter en la salida estándar, con la peculiaridad de que es capaz de sustituir variables por sus valores. Por ejemplo, si se ha asignado la variable `mensaje=hola`, posteriormente se podrá utilizar en cualquier comando o simplemente visualizarla en pantalla: `echo "$mensaje mundo"` imprimirá en la salida estándar el texto `"hola mundo"`.

```
usuario@homeless-flight:~$ test='ls -lt'
usuario@homeless-flight:~$ $test
total 12
--wx-wx--x 2 usuario usuario    6 Feb 18 18:11 prueba2.txt
--wx-wx--x 2 usuario usuario    6 Feb 18 18:11 prueba.txt
drwxrwxr-x 2 usuario usuario 4096 Feb 18 17:57 prueba
usuario@homeless-flight:~$
```

Figura 75. Ejemplo de variable.

Fuente: elaboración propia.

En el ejemplo, se define la variable **test="ls -l"**, que contendrá el texto de un comando ejecutable, y a continuación se invoca con **\$test** (figura 75).

Lo que ha sucedido al escribir **\$test** en la consola es que se ha sustituido la variable **test** por su contenido. En este caso, el contenido era un comando y, al pulsar **Enter**, se ha interpretado y ejecutado.

```

usuario@homeless-flight:~$ test='ls -lt'
usuario@homeless-flight:~$ $test
total 12
--wx-wx--x 2 usuario usuario    6 Feb 18 18:11 prueba2.txt
--wx-wx--x 2 usuario usuario    6 Feb 18 18:11 prueba.txt
drwxrwxr-x 2 usuario usuario 4096 Feb 18 17:57 prueba
usuario@homeless-flight:~$ unset test
usuario@homeless-flight:~$ $test
usuario@homeless-flight:~$ █

```

Figura 76. Eliminación de una variable.

Fuente: elaboración propia.

Para eliminar una variable, se usa el comando **unset**. Por ejemplo (figura 76).

Existen algunas variables definidas en la *shell*. En la siguiente tabla se muestran las principales variables (tabla 2):

Nombre de la variable	Significado
LOGNAME	Nombre del usuario.
HOME	Directorio de trabajo del usuario actual.
PATH	Caminos usados para ejecutar órdenes o programas.
PWD	Directorio activo.
TERM	El tipo de terminal actual.
SHELL	<i>Shell</i> actual.
\$?	Contiene el resultado de la ejecución del comando anterior. El valor '0' indica que no ha habido errores, mientras que otros valores indican errores.

Tabla 2. Algunas variables predefinidas.

Fuente: elaboración propia.

CONTINUAR

Un script puede recibir parámetros en la línea de órdenes para considerarlos durante su ejecución. Los parámetros recibidos se guardan en una serie de variables especiales que el script puede consultar cuando lo necesite. Los nombres de estas variables son: \$1 \$2 \$3 ... \${10} \${11} \${12} ... de manera que:

- La variable \$0 contiene el nombre con el que se ha invocado el script, es decir, el nombre del programa.
- \$1 contiene el primer parámetro.
- \$2 contiene el segundo parámetro.
- ...

A continuación, en el siguiente ejemplo de un script *shell*, se muestran los valores de los parámetros (figura 77):

```
echo el nombre del programa es $0
echo el primer parámetro es $1
echo el segundo parámetro es $2
```

```
usuario@homeless-flight:~$ bash script parámetro1 parámetro2
el nombre del programa es script
el primer parámetro es parámetro1
el segundo parámetro es parámetro2
usuario@homeless-flight:~$
```

Figura 77. Ejemplo de usos de parámetros.

Fuente: elaboración propia.

CONTINUAR

La orden **shift** mueve todos los parámetros una posición a la izquierda, esto hace que el contenido del parámetro \$1 desaparezca y sea reemplazado por el contenido de \$2, \$2, que es reemplazado por \$3, etc. Además, existen algunas variables especiales como \$# , que contiene el número de parámetros que ha recibido el script, y \${*} o \${@}, que contiene todos los parámetros recibidos.

A continuación, se describen algunas **reglas para el uso y evaluación de variables**:

\$var o \${var} —

Representa el valor de la variable **var** o nada si la variable no está definida.

`\${var-\$val}` —

Devuelve el valor de **var** si está definida, en caso de no estarlo, se devolvería el valor contenido en la variable **val**.

`\${var=val}` —

Valor de **var** si está definida. En caso de que **var** no esté definida, se le asigna el valor de **val** y se retorna su contenido.

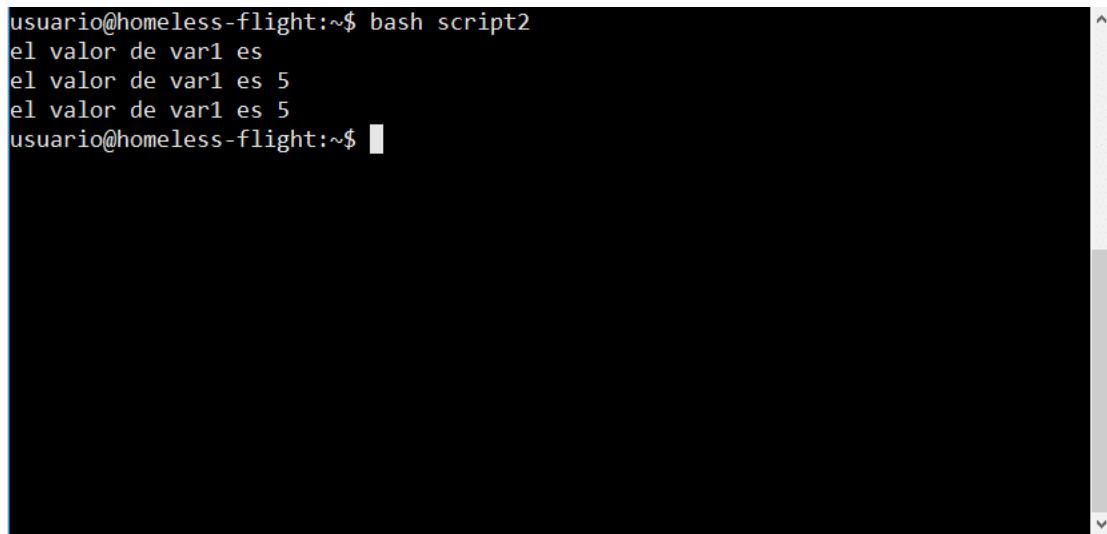
suma=\$((\$var1 + \$var2 + \$var3...)) —

Se pueden englobar operaciones aritméticas dentro de una secuencia \$((...)). En este ejemplo, se evalúan los contenidos de **var1**, **var2** y **var3**, se sumarían aquellos valores que fueran numéricos (los no numéricos se descartarían) y se guardaría el resultado en la nueva variable **suma**.

El siguiente ejemplo muestra cómo usar una variable asignándole un valor en caso de que no esté definida (figura 78):

Nombre de archivo: **script2**

```
echo el valor de var1 es $var1
echo el valor de var1 es ${var1=5}
echo el valor de var1 es $var1
```



```
usuario@homeless-flight:~$ bash script2
el valor de var1 es
el valor de var1 es 5
el valor de var1 es 5
usuario@homeless-flight:~$
```

Figura 78. Ejemplo de uso de parámetros.
Fuente: elaboración propia.

A continuación, en el siguiente script *shell*, se muestra el directorio actual y el usuario logeado.

Nombre de archivo: *script3*

```
#!/bin/bash
echo "El directorio actual es:"
pwd
echo "Usuario logeado:"
#Comando que muestra el usuario que lo ejecuta
whoami
```

CONTINUAR

Comillas dobles y simples

Pueden usarse comillas simples o dobles cuando se asignan caracteres que contengan espacios y caracteres especiales. Sin embargo, forzosamente deben usarse comillas dobles si se quiere utilizar variables dentro de textos entrecomillados.

El siguiente script *shell* **concatena** el valor de una variable con otra.

```
#!/bin/bash
var="Hola Linux"
var2="$var y Scripts de Shell"
echo $var2
```

Conclusión:

La salida de este script será el texto: **Hola Linux y Scripts de Shell.**

Usando comillas simples, el siguiente script *shell* **no concatena** el valor de una variable con otra.

```
#!/bin/bash
var='Hola Linux'
var2='$var y Scripts de Shell'
echo $var2
```

Conclusión:

La salida de este script será el texto: **\$var y Scripts de Shell.**

CONTINUAR

Arrays

Un array **es una colección de elementos del mismo tipo que se almacenan en posiciones contiguas de memoria.** El primer elemento del array está en la posición 0 del array, el segundo en la 1, etc. No hay un tamaño límite para un array y la asignación de valores se puede hacer de forma alterna, accediendo directamente a posiciones concretas. La sintaxis para declarar un array es la siguiente:

- Crear e iniciar un array: **nombre_array=(val1 val2 val3 ...)**.
- Asignar un valor al elemento en la posición 2 del array: **nombre_array[2]=valor**.
- Acceder al elemento en la posición 0: **\${nombre_array[0]}**.
- Consultar todos los elementos: **\${nombre_array[*]} o \${nombre_array[@]}**.

Atención:

Es preciso tener en cuenta lo siguiente:

Si al referenciar un array no se utiliza subíndice (**nombre_array[posición]**), se considera que se está referenciando su primer elemento.

A continuación, el siguiente script define dos arrays, que son animales mamíferos y animales ovíparos:

```
#!/bin/bash
mamiferos=('perro' 'gato' 'león' 'mono')
oviparos=("loro" "gallina" "pato" "ganso" "abeja")
#imprimir perro
echo ${mamiferos[0]}
#imprimir gato
echo ${mamiferos[1]}
#imprimir león
echo ${mamiferos[2]}
#imprimir todos los animales ovíparos
echo ${oviparos[*]}
```

CONTINUAR

Órdenes internas de la shell

Una orden interna de la shell **es una orden que el intérprete implementa y que ejecuta sin llamar a programas externos.**



echo: envía una cadena a la salida estándar. Por ejemplo:

```
#!/bin/bash
echo Esto es una cadena
```



read: lee una cadena de la entrada estándar. Por ejemplo:

```
#!/bin/bash
echo -n "Introduzca un valor para var1: "
read var1
echo "var1 = $var1"
```

La orden **read** puede leer diferentes variables a la vez. También se puede combinar el uso de **read** con **echo** para mostrar un prompt que indique qué es lo que se está pidiendo. Hay una serie de caracteres especiales para usar en **echo** y que permiten colocar el cursor en un sitio determinado:

\b: retrocede una posición (sin borrar).

\f: alimentación de página.

\n: salto de línea.

\t: tabulador.

Para que **echo** reconozca estos caracteres es necesario utilizar la opción **-e** y comillas dobles:

```
$ echo -e "Hola \t ¿cómo estás?"
```

EXIT

TRUE Y FALSE

Finaliza la ejecución del script, recibe como argumento un entero que será el valor de retorno.

EXIT

TRUE Y FALSE

Devuelven 0 y 1 siempre, respectivamente. El valor 0 se corresponde con true y cualquier valor distinto de 0 con **false**.

CONTINUAR

Estructuras de control

IF y CASE

Permiten limitar la ejecución de ciertos comandos u otras sentencias al cumplimiento de unas condiciones definidas, generalmente en **expresiones**.

Las expresiones se engloban entre los caracteres [y], los cuales deben estar separados de la expresión por un espacio cada uno.

Cada expresión se puede evaluar como **true** o **false**, según se cumpla o no su definición. Normalmente se utilizan expresiones para comparar variables, aunque se puede extender su uso a casos más complejos, como evaluar la existencia o no de un fichero en el sistema.

La sintaxis permite utilizar operadores **unarios** (que evalúan una variable o literal) o **binarios** (que evalúan dos variables o literales):

- **[op_unario \$var]** evalúa si el contenido de \$var cumple las condiciones del operador.
- **[\$var1 op \$var2]** evalúa si ambas variables cumplen con las condiciones definidas por el operador.

Algunos de los comparadores más frecuentes son:¹⁶

¹⁶[Bash Test Operators Cheatsheet](#).

[\$var1 -eq \$var2]

Si ambas variables contienen valores numéricos, la expresión será **true** si ambas variables contienen el mismo valor. En otro caso será **false**.

[\$var1 -lt \$var2]

Si ambas variables contienen valores numéricos, la expresión será **true** si el contenido de var1 es menor que el de var2. En otro caso será **false**.

[\$var1 -le \$var2]

Si ambas variables contienen valores numéricos, la expresión será **true** si el contenido de var1 es menor o igual que el de var2. En otro caso será **false**.

[\$var1 -gt \$var2]

Si ambas variables contienen valores numéricos, la expresión será **true** si el contenido de var1 es mayor que el de var2. En otro caso será **false**.

[\$var1 -ge \$var2]

Si ambas variables contienen valores numéricos, la expresión será **true** si el contenido de var1 es mayor o igual que el de var2. En otro caso será **false**.

[“\$var1” = “\$var2”]

Si ambas variables contienen valores de texto, la expresión será **true** si el contenido de var1 es igual que el de var2. En otro caso será **false**.

Saber más

Más adelante se verá el comando **test**, que permite evaluar este tipo de expresiones, y se explicarán en detalle otros operadores disponibles.

CONTINUAR

Estructura if



Define la condición explícita bajo la cual se deben ejecutar una o varias sentencias.

```
if [expresión]
then
# órdenes a ejecutar si se cumple la condición definida en expresión
    sentencia1
    sentencia2
    ...
elif [expresión]
then
# órdenes a ejecutar si se cumple la condición de la expresión definida en elif
```

```
sentencia3
sentencia4
...
else
# órdenes a ejecutar en cualquier otro caso
    sentencia5
    sentencia6
...
fi
```

Ejemplo básico de la sentencia if, script que verifica si el directorio actual es "/root":

```
#!/bin/bash
directorio=$(pwd)      #Asiga la salida del comando pwd a la variable directorio
if [ "$directorio" = "/root" ]; then
    echo "Directorio actual es el directorio /root"
else
    echo "Directorio actual diferente de: /root"
fi
```

El siguiente script pregunta cuál es el nombre del usuario actual y lo valida:

```
#!/bin/bash
echo -n "Escribe tu nombre de usuario: "
read usuario
if [ "$usuario" = "$USER" ]; then
    echo "Buen día, $usuario."
else
    echo "¡No eres el usuario $usuario!"
fi
```

CONTINUAR

Estructura case



Permite definir conjuntos de condiciones y las sentencias que se ejecutarán cuando se cumpla cada condición.

Cada conjunto de sentencias debe terminar con los caracteres especiales ;;

```
case $var in
v1) ... #Acciones a realizar si var toma el valor v1
;;
v2|v3) ...#Acciones a realizar si var toma el valor v2 o v3
;;
*) ...# Caso por defecto si no se cumple ninguna de las anteriores
;;
esac
```

Por ejemplo, el siguiente script solicita un número e imprime el valor con caracteres:

```
#!/bin/bash
echo -n "Escribe un número entre 1 y 5: "
read x
case $x in
    1) echo "Tecleaste el número uno.";;
    2) echo "Tecleaste el número dos.";;
    3) echo "Tecleaste el número tres.";;
    4) echo "Tecleaste el número cuatro.";;
    5) echo "Tecleaste el número cinco.";;
    *) echo "Error, debías escribir un número entre 1 y 5";;
esac
```

[CONTINUAR](#)

BUCLES

Los bucles **permiten ejecutar grupos de sentencias de forma repetida**, mientras se satisfagan las condiciones definidas.

Estructura while

Esta estructura encapsula un conjunto de sentencias que debe ejecutarse mientras se cumpla la condición definida en una **expresión**: [expresión]

```
while [ expresión ] # Mientras la expresión sea cierta ...
do
    sentencia1
    sentencia2
    ...
done
```

Por ejemplo, **un script que lee del teclado un número y lo suma con sus antecesores**:

```
#!/bin/bash
echo -n "Escribe un número: "
read numero
suma=0
contador=1
while [ $contador -le $numero ]; do # le -> less than
    suma=$((suma + $contador))
    contador=$((contador + 1))
done
echo "La suma del 1 al $numero es: $suma"
```

Otro ejemplo sería el de **un script que calcula la factorial de un número usando el bucle while**, recibe el número como parámetro:

```
#!/bin/bash
contador=$1
numero=$contador
factorial=1
while [ $contador -gt 0 ] # gt -> greater than
do
    factorial=$(( $factorial * $contador ))
    contador=$(( $contador - 1 ))
done
echo "El factorial de $numero es $factorial"
```

CONTINUAR

Estructura until

El bucle until es similar al bucle while, la diferencia es que **el código se ejecuta mientras la expresión se evalúe como falsa**.

```
until [ expresión ] # Mientras la expresión sea falsa ...
do
...
Done
```

Por ejemplo, un script que imprime cinco veces el mensaje "Hola mundo":

```
#!/bin/bash
contador=1
until [ $contador -gt 5 ] # gt -> greater than
do
    echo "Hola mundo $contador vez."
    contador=$(( $contador+1 ))
done
```

CONTINUAR

Estructura for

Recorre un array y, para cada elemento, **ejecuta un bloque de sentencias**.

```
for var in lista #Por cada valor en el array se ejecuta una iteración.
do
...órdenes a ejecutar
Done
```

Por ejemplo:

```
for i in 10 30 70
do
    echo "Mi número favorito es $i"
done
```

O un script que imprime cadenas utilizando un bucle for:

```
#!/bin/bash
for x in Cabeza cara orejas ojos nariz boca; do
    echo "El valor de la variable x es: $x"
    sleep 1 # Se pausa la ejecución durante 1 segundo
done
```

CONTINUAR

Break y continue

Sirven para **interrumpir la ejecución secuencial del cuerpo del bucle**:



break transfiere el control a la orden que sigue a **done**, haciendo que el bucle termine antes de tiempo.

Ejemplo: bucle while que **imprime del número 0 al 9 utilizando break se termina el ciclo si el contador es igual a 5**:

```
#!/bin/bash
contador=0
while [ $contador -lt 10 ]
do
    echo $contador
    if [ $contador -eq 5 ]
    then
        break
    fi
    contador=`expr $contador + 1`
done
```



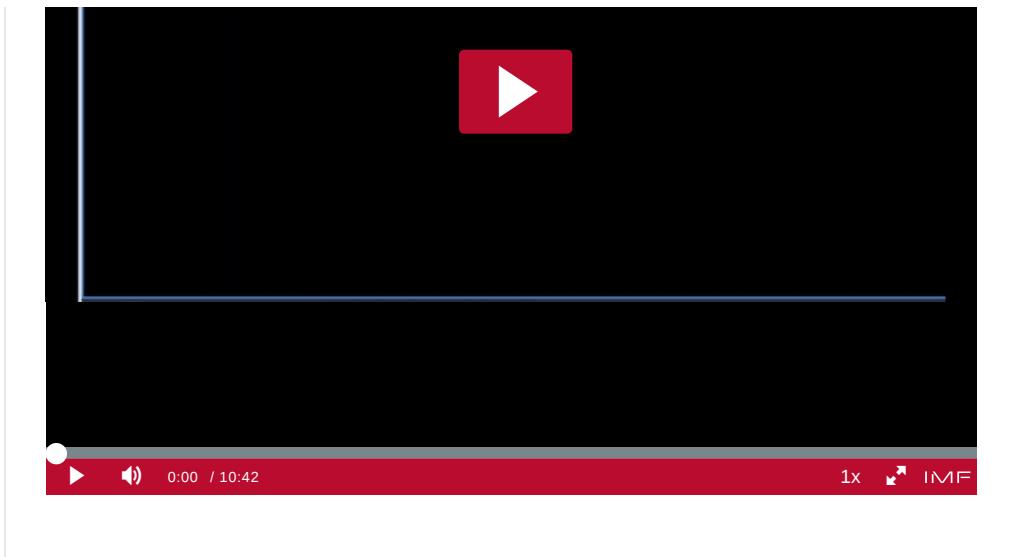
continue transfiere el control a **done**, haciendo que se evalúe de nuevo la condición, prosiguiendo el bucle.

Por ejemplo, un script que **lee un archivo y solo imprime las líneas con menos de 20 caracteres**:

```
#!/bin/bash
while read buf
do
    cuenta=`echo $buf | wc -c`
    if [ $cuenta -gt 20 ]
    then
        continue
    fi
    echo $buf
done < $1
```

Anotación:

En ambos casos, las órdenes del cuerpo del bucle siguientes a estas sentencias no se ejecutan. Lo normal es que formen parte de una sentencia condicional, como **if**.



CONTINUAR

La orden test

test es una orden que permite evaluar si una expresión es verdadera o falsa.

Se puede usar como alternativa a la sintaxis de evaluación de expresiones vista hasta ahora: [expresión] o **test** expresión son totalmente equivalentes y ambas se pueden utilizar en las sentencias de flujo condicional vistas hasta ahora: **if/while/until**.

Algunos operadores que pueden aparecer en las condiciones evaluadas:

- **Operadores para números** (tabla 3):

-eq	Igual a
-ne	Distinto de
-lt	Menor que
-le	Menor o igual que
-gt	Mayor que

-ge	Mayor o igual que
Tabla 3. Operadores para números. Fuente: elaboración propia.	

Por ejemplo:

```
if [ $e -eq 1 ] # ó if test $e -eq 1
then
echo Vale 1
else
echo No vale 1
fi
```

- **Operadores para cadenas (tabla 4):**

-z cadena	Verdad si la longitud de cadena es cero.
-n cadena o cadena	Verdad si la longitud de cadena no es cero.
cadena1 == cadena2	Verdad si las cadenas son iguales. Se puede emplear = en vez de ==.
cadena1 != cadena2	Verdad si las cadenas no son iguales.
cadena1 < cadena2	Verdad si cadena1 se ordena lexicográficamente antes de cadena2 en la localización en curso.
cadena1 > cadena2	Verdad si cadena1 se ordena lexicográficamente después de cadena2 en la localización en curso.
Tabla 4. Operadores para cadenas. Fuente: elaboración propia.	

Por ejemplo:

```
#!/bin/bash
S1='cadena'
S2='Cadena'
if [ $S1!=$S2 ]; then
echo "S1(''$S1') no es igual a S2(''$S2')"
fi
if [ $S1==$S1 ]; then
echo "S1(''$S1') es igual a S1(''$S1')"
fi
```

- Operadores sobre ficheros (tabla 5):

-e fichero	El fichero existe.
-r fichero	El fichero existe y se tiene permiso de lectura.
-w fichero	El fichero existe y se tiene permiso de escritura.
-x fichero	El fichero existe y se tiene permiso de ejecución.
-f fichero	El fichero existe y es regular.
-s fichero	El fichero existe y es de tamaño mayor a cero.
-d fichero	El fichero existe y es un directorio.

Tabla 5. Operadores sobre ficheros.
Fuente: elaboración propia.

Por ejemplo:

```
if test -f "$1" # ¿ es un fichero ?
then
more $1
elif test -d "$1" # ¿ es un directorio ?
then
(cd $1;ls -l|more)
else # no es ni fichero ni directorio
echo "$1 no es fichero ni directorio"
fi
```

- Operadores lógicos (tabla 6):

-o	OR
-a	AND
!	NOT
\(Paréntesis izquierdo

\)	Paréntesis derecho
Tabla 6. Operadores lógicos. Fuente: elaboración propia.	

CONTINUAR

Funciones

Las funciones **permiten agrupar bloques de sentencias que se pueden reutilizar en cualquier parte de un script o guion.**

Para declarar una función:

```
function nombre{
...mi_código
}
```



Llamar a la función es como llamar a otro programa o comando, sólo hay que escribir su nombre.

Por ejemplo:

```
# Se define la función hola
function hola {
echo ¡Hola!
}

hola # Se llama a la función hola
```

Las funciones pueden recibir parámetros, que serán tratados de forma similar a como se tratan los parámetros de entrada a un script.

```
#!/bin/bash
function saludar {
echo $1
}
```

Se podría invocar como:

```
# una vez dentro de la función saludar, $1 contiene el valor "Hola"  
saludar Hola
```



CONTINUAR

Algunos ejemplos más de scripts

Ejemplo 1

Script que solicita confirmación si va a sobrescribir un archivo cuando se use la orden **cp**.

```
#!/bin/bash
if [ -f $2 ]
then
    echo "$2 existe. ¿Quieres sobreescribirlo? (s/n)"
    read sn
    if [ $sn = "N"-o $sn = "n" ]
    then
        exit 0 # para la ejecución del script
    fi
fi
cp $1 $2.
```

Ejemplo 2

Script que liste los directorios existentes en el directorio actual.

```
#!/bin/bash
for archivo in *
do
    test -d $archivo && ls $archivo
    # && permite la ejecución del siguiente comando si el anterior tuvo éxito.
done
```

Ejemplo 3

Script que borra con confirmación todos los archivos indicados como argumentos en la línea de comandos.

```
#!/bin/bash
hwhile test "$1" != ""
do
    rm -i $1
    shift
done
```

Ejemplo 4

Script que evalúa la extensión de un archivo. Si esta se corresponde con .txt., copia el archivo al directorio “~/copias”. Si es otra la extensión, presenta un mensaje.

```
case $1 in
*.txt)
    cp $1 ~/copias/$1
;*
*)
    echo "$1 extensión desconocida"
;*
esac
```

Ejemplo 5

Script que pone el atributo de ejecutable a los archivos pasados como argumento.

```
for fich in $@
do
    if test -f $fich
    then
        chmod u+x $fich
    fi
done
```

Ejemplo 6

Script que lee dos números del teclado e imprime su suma.

```
#!/bin/bash
echo "Introduzca un número \n"
read numero1
echo "Introduzca otro número \n"
read numero2

respuesta=$((numero1 + numero2))
echo "$numero1 + $numero2 = $respuesta \n"
```

Ejemplo 7

Script que lista los directorios existentes en el directorio "/root".

```
#!/bin/bash
# Lista todos los ficheros del directorio /root
for fichero in /root
do
    ls -l "$fichero"
done
```

Ejemplo 8

Script que lista los enlaces simbólicos del directorio que se indique.

```
#!/bin/bash
hecho "Escribe el nombre del directorio: "
read directorio
echo "Enlaces simbolicos en el directorio $directorio"
for fichero in $( find $directorio -type l )
do
    echo "$fichero"
done
```

Ejemplo 9

Usando funciones, script que verifica si existe un fichero en el directorio “/root”.

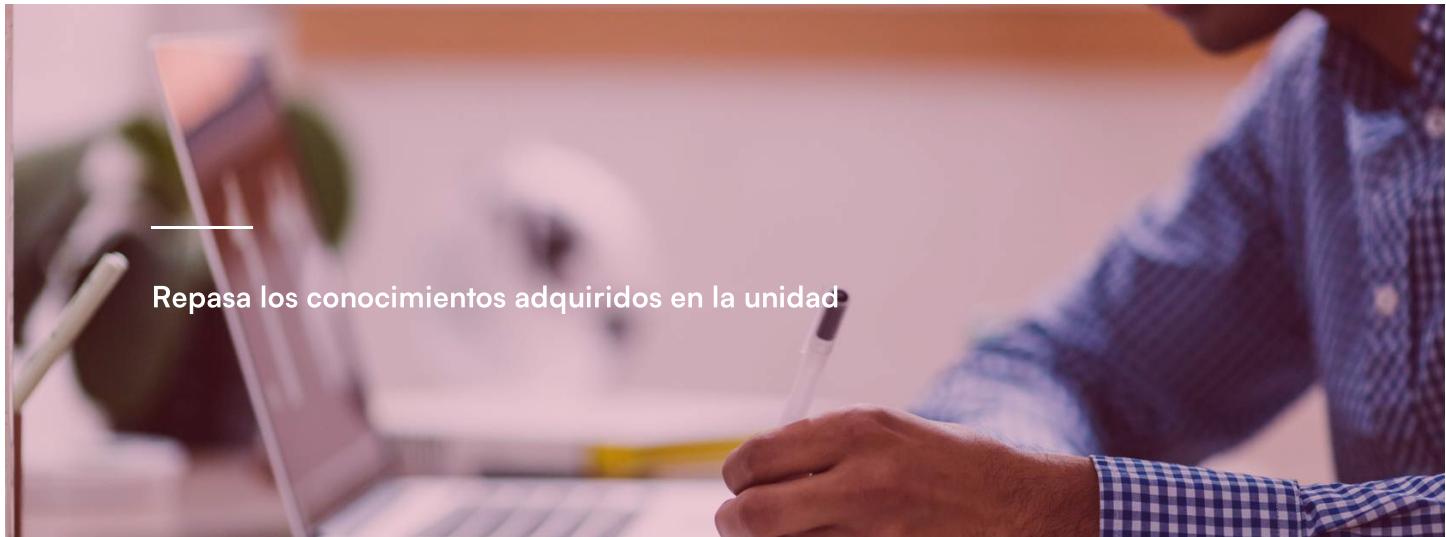
```
#!/bin/bash
hfunction check() {
    if [-e "/root/$1"]
    then
        return 0
    else
        return 1
    fi
}
echo "Introduzca el nombre del archivo:"
read fichero
if check $fichero
    then
        echo "El fichero $fichero existe."
    else
        echo "El fichero $fichero no existe."
    fi
```

Ejemplo 10

Script que recibe como parámetro un nombre de usuario y verifica si existe en el sistema operativo.

```
#!/bin/bash
# Parámetros
usuario=$1
es_usuario=`grep $usuario /etc/passwd`
if test -z "$es_usuario"
then
    echo "$usuario no es usuario del sistema operativo"
else
    echo "$usuario si es usuario del sistema operativo"
fi
```

VII. Resumen



Repasa los conocimientos adquiridos en la unidad

En esta unidad, se ha introducido el concepto de máquina virtual como un mecanismo que facilita la ejecución de una simulación de un sistema operativo diferente al que se tiene instalado por defecto. Para poder crear y ejecutar una máquina virtual, se requiere un software específico. Las dos herramientas más importantes para este fin son Virtual Box y VMware.

En esta unidad se ha descrito cómo usar la herramienta Virtual Box para crear, ejecutar y configurar máquinas virtuales. Para ello, se ha creado una máquina virtual a partir de una distribución de Linux denominada Lubuntu.

Asimismo, se ha descrito cómo se puede importar una máquina virtual creada por terceros. Para este caso, se ha utilizado una máquina virtual de una distribución de Linux de tipo Lubuntu.

Una vez se ha podido ejecutar un sistema operativo como Linux en un entorno Windows, se ha accedido a un terminal de comandos donde aprender a utilizar una *shell* de comandos, en este caso, la *shell* más popular entre los sistemas Linux actuales: BourneAgain Shell (Bash).

Bash no es únicamente una interfaz textual donde introducir comandos y obtener resultados. Bash ofrece un potente lenguaje de scripting con estructuras de control semejantes a las de cualquier otro lenguaje de programación y da la flexibilidad necesaria para componer sistemas automáticos para el procesamiento de datos.

Dentro de la *shell*, el alumno ha aprendido primero a navegar por el sistema de ficheros y a entender el sistema de asignación de permisos dentro de este nuevo entorno multiusuario.

Se han repasado los comandos básicos que permiten manipular el sistema de ficheros: **cp**, **mv**, **ls**, **ln...**, así como los comandos más populares para el procesamiento de ficheros de texto, filtrado y extracción de información útil a partir de ellos: **cat**, **grep**, **sed...**

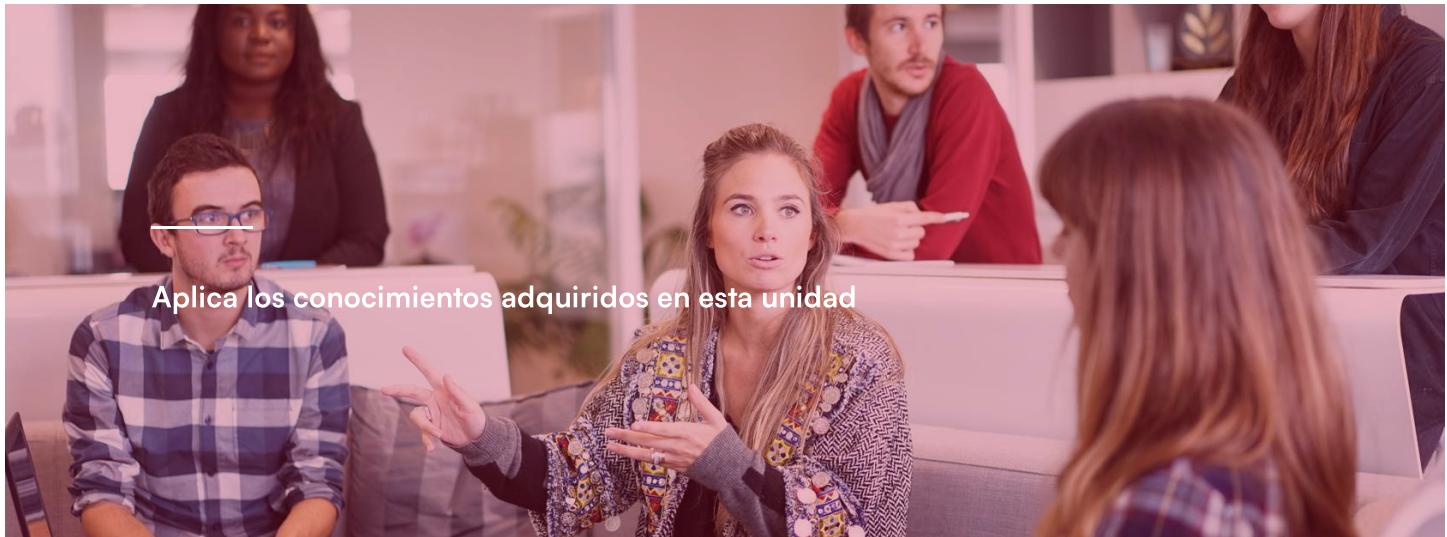
Posteriormente se ha profundizado en las estructuras de control que ofrece el lenguaje de scripting de Bash y se ha aprendido a automatizar la ejecución de comandos en función de determinadas condiciones.

Por último, se ha explicado cómo construir guiones o scripts con algoritmos complejos que permitan definir los automatismos necesarios para el día a día. La *shell* permite construir programas propios para ejecutarlos con la frecuencia deseada abstrayéndose de la complejidad subyacente.

En definitiva, dominar la *shell* de Bash permite a los analistas de datos o científicos de datos acceder directamente a los sistemas donde se genera la información (mayoritariamente basados en Linux), procesarla, filtrarla y transformarla.

Esto permitirá bien obtener datos de interés, componer agregados para su posterior uso en otros sistemas o incluso volcarlos a una base de datos para un análisis posterior más profundo.

VIII. Caso práctico con solución



Aplica los conocimientos adquiridos en esta unidad

ENUNCIADO

Se pide crear un script que muestre el siguiente menú:

Ver directorio actual.....[1]
Copiar archivos.....[2]
Editar archivos.....[3]
Imprimir archivo.....[4]
Salir del menú.....[5]

DATOS

- Si elige la primera opción, se mostrarán los archivos del directorio actual.
- Si elige la segunda opción, se le pedirá el nombre del archivo que se quiere copiar y el nombre del archivo donde se quiere copiar.
- Si elige la tercera opción, se le pedirá el nombre del archivo que se desea editar y se abrirá el editor vi para editarlo.
- Si elige la cuarta opción, se le pedirá el nombre del archivo que se desea imprimir y este se imprimirá.
- Si elige la quinta opción, se saldrá del script.

¿Cuál sería el script?

[VER SOLUCIÓN](#)

SOLUCIÓN

```
while true
do
clear
echo "
Ver directorio actual.....[1]
Copiar ficheros.....[2]
Editar ficheros.....[3]
Imprimir fichero.....[4]
Salir del menú.....[5]"
read i
case $i in
1) ls -l|more; read z
;;
2) echo "Introduzca [desde] [hasta]"
read x y
cp $x $y
read x
;;
3) echo "¿Nombre de fichero a editar?"
read x;
vi $x
;;
4) echo "¿Nombre de fichero a imprimir?"
read x
lpr $x
;;
5) clear; break
;;
esac
done
```

IX. Lecturas recomendadas

Lecturas recomendadas

En esta sección podréis acceder a las lecturas recomendadas para esta unidad.

- VirtualBox.org. [Manual oficial de Virtual Box](#).
- Linuxconfig.org. [Tutorial de Bash](#).

X. Glosario



El glosario contiene términos destacados para la comprensión de la unidad

Echo —

Este comando se utiliza para mostrar en pantalla (en terminal) una cadena de texto que se le pasa como argumento; es decir, su uso principal es el de mostrar texto en pantalla durante la ejecución de scripts.

Máquina virtual —

Es una simulación de un sistema operativo. Físicamente es un archivo con varias extensiones posibles, como .ova, .vmdk, etc.

Sistema operativo anfitrión —

Sistema operativo sobre el que se ejecuta una máquina virtual.

Sistema operativo huésped —

Sistema operativo virtualizado que se ejecuta sobre el sistema operativo anfitrión.

Virtual Box —

Software gratuito para crear máquinas virtuales.

VMware —

Software con diferentes versiones de licencias para crear máquinas virtuales.

Sistema operativo virtualizado —

Es un sinónimo de máquina virtual.

Software multiplataforma —

Es un software que puede ejecutarse en diferentes tipos de sistemas operativos (Windows, Linux, Mac, etc.).

Lubuntu —

Es una distribución de Linux que requiere pocos recursos hardware.

Virtualizar —

Es la acción de crear una máquina virtual.

Crear una máquina virtual —

Es un sinónimo de virtualizar.

Importar/cargar máquina virtual —

Es la acción de importar una máquina virtual que ha sido creada por un tercero a un sistema de gestión de máquinas virtuales para poder usarla.

Exportar una máquina virtual —

Es la acción de exportar una máquina virtual creada para distribuirla entre otras personas.

Shell o intérprete —

:Aplicación que permite al usuario interactuar con el sistema operativo mediante la ejecución de comandos o sentencias de texto.

Script o guion —

Es un archivo de texto que contiene comandos de la **shell** que se interpretarán cuando se ejecute el script.

Apagado ACPI —

Es una señal hardware, normalmente generada por el botón **Power** de un ordenador, para indicar al sistema operativo que debe apagarse de forma ordenada.

XI. Bibliografía

- Garrels, M. [Bash Guide for Beginners](#). 2008.
- Guía oficial de [primeros pasos con VirtualBox](#).
- [User manual](#).
- Love, R. *Linux Kernel Development*. Addison Wesley; 2010, 3.^a ed.
- Matthew, N. y Stones, R. *Beginning Linux Programming*. Ed. Wiley; 2007, 4.^a ed.
- Mitchell, M. et al. [Advanced Linux Programming](#). Indianapolis, IN: New Riders Publishing; 2001.
- Kerrisk, M. *The Linux Programming Interface*. No Starch Press; 2010.