

Capítulo 2

Programación de sistemas operativos. Lenguajes y librerías de uso común

Contenido

1. Introducción
2. Las llamadas al sistema (*system calls*)
3. Programas de utilidades y comandos del sistema
4. Resumen

1. Introducción

Los sistemas operativos, además de ofrecer innumerables servicios para los usuarios, también proporcionan las herramientas necesarias para el desarrollo de todo tipo de aplicaciones con el fin de que estas hagan uso del sistema tal cual lo concibe el usuario.

Cuando el usuario ejecuta una aplicación en un sistema operativo moderno, se producen miles de operaciones que preparan el entorno que necesita la aplicación para poder usar todos los recursos que el sistema pone a su disposición.

Sin embargo, este tipo de llamadas no solo se producen en el momento de la ejecución de la aplicación, sino que lo hacen de forma continua durante todo el tiempo que la aplicación está desplegada en el sistema. Estas llamadas pertenecen a un conjunto de funciones que residen en el corazón del sistema operativo y por sí mismas; se podría decir que son lo que debería considerarse como el propio sistema operativo. Si se pudiera trazar una raya para determinar qué es en realidad un sistema operativo, sería en ese punto (todo lo que quedase a un lado de estas llamadas no debería considerarse parte del sistema operativo).

2. Las llamadas al sistema (*system calls*)

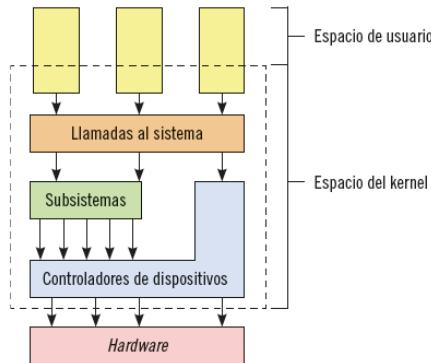
En los procesadores modernos existen dos modos de funcionamiento o ejecución. Estos son el modo usuario y el modo *kernel* o protegido.

En el **modo usuario** se ejecutan los procesos y bibliotecas del sistema operativo; solo existe un conjunto de instrucciones disponibles para su ejecución y, cuando se realiza una llamada a una instrucción que no pertenece a este modo, el sistema operativo ignora la llamada o bien lanza una excepción.

En el **modo kernel** o modo protegido no existen restricciones; a través de él se accede de forma directa al *hardware*. Todo el núcleo del sistema operativo se ejecuta en este modo y las llamadas al sistema pertenecen a este modo.

Cuando un proceso realiza una llamada al sistema, el sistema operativo realiza un cambio de contexto. Esto significa que la ejecución pasa del modo usuario al modo *kernel*, guardando el estado del proceso que realizó la llamada para que, cuando finalice, sea restaurado y siga su ejecución tras la llamada al sistema.

Diagrama de bloques funcionales de un sistema operativo



Actividades

1. ¿Podría encontrarse el código de una llamada al sistema en una aplicación que se ha bajado de internet? Razona la respuesta.

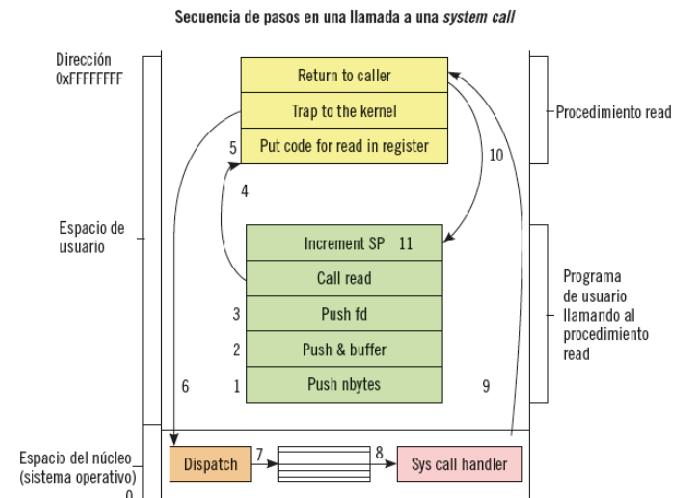
2.1. Definición

Las **llamadas al sistema** (*system calls*) son instrucciones que forman parte del núcleo del sistema operativo. Su ejecución se realiza en modo *kernel*, es decir, un modo restringido de ejecución, y actúan como una interfaz entre los programas del espacio de usuario y los servicios que proporciona el sistema operativo. Definen todas las acciones que el sistema operativo puede realizar.

La implementación y el conjunto de todas las llamadas al sistema pueden variar dependiendo del sistema operativo. Algunas son implementadas en ensamblador y otras en lenguajes de alto nivel, generalmente en C, Java, Perl, Python, etc.

Se ponen a disposición del programador como funciones de bibliotecas, aunque realmente estas funciones son envoltorios que usan las propias llamadas al sistema. En general, el nombre de estas funciones coincide con el nombre de la llamada al sistema que envuelven.

Ejemplo de ejecución de una llamada al sistema usando una biblioteca: fread(fd, buffer, nbytes).



El manejador mantiene una tabla de funciones que son usadas para cada llamada al sistema. Tras finalizar la llamada al sistema, el manejador devuelve el control de ejecución al proceso.

La secuencia de pasos que se produce cuando se realiza una llamada al sistema es:

- 1, 2 y 3. El proceso de usuario introduce en la pila los valores de los parámetros de la función que envuelve la llamada al sistema.
- 4. Después realiza la llamada a la función.
- 5. Esa función coloca en el registro del procesador el número que identifica la llamada específica al sistema.
- 6. Después, se ejecuta la interrupción o *trap*, que provoca la ejecución en modo *kernel*.
- 7. El *kernel* devuelve al manejador de la llamada al sistema el código almacenado de esa llamada, que corresponde al número que se encuentra en el registro del procesador.
- 8. Se ejecuta la llamada al sistema.
- 9. Cuando acaba la ejecución, el manejador devuelve el control de ejecución al proceso que realiza la llamada. Conmutando al modo usuario.
- 10. Acaba la ejecución de la función que envuelve la llamada al sistema.
- 11. Se actualiza la pila de ejecución al estado anterior a la llamada.

En resumen, las llamadas al sistema se utilizan para comunicarse con el *kernel* del SO y brindan una forma de obtener recursos, información, etc. En la actualidad, nada se puede hacer sin la ayuda del SO y, por lo tanto, un proceso depende bastante de esas llamadas al sistema. En concreto, el procesador cambia de modo usuario a modo *kernel* varias veces por segundo.



Sabía que...

Los sistemas operativos actuales ejecutan cientos de llamadas a sistema distintas.



Actividades

2. ¿Por qué cree que es tan necesaria la existencia de dos modos de ejecución (usuario y *kernel*) para el sistema operativo?

2.2. Uso directo y mediante Application Programming Interfaces (API)

En el apartado anterior se ha visto cómo usar una llamada al sistema utilizando una función que envolvía la llamada. Las llamadas al sistema se pueden realizar de forma directa, usando mecanismos que pone a disposición el sistema operativo, o bien usando una API (*Application Programming Interface*).

Una API es una interfaz de programación de aplicaciones que se utiliza para el desarrollo. Es un conjunto de funciones agrupadas en bibliotecas que se proporcionan al programador para facilitar el desarrollo de las aplicaciones.

Existen APIs específicas que agrupan las funciones que hacen uso de las llamadas al sistema. Las llamadas del sistema difieren de una plataforma a otra, pero usando una API estable es fácil migrar el *software* entre diferentes plataformas.

Las tres grandes API que envuelven llamadas al sistema son: *Win32 API* para Windows, *POSIX API* para sistemas basados en *POSIX* (incluyendo todas las versiones *Unix*, *Linux* y *macOS*), y *Java API* para la máquina virtual de *Java*.

Un ejemplo de llamada al sistema de forma directa en un sistema operativo basado en *Linux* sin usar API es:

```
include <syscall.h>
extern int syscall(int, ...);
int my_close(int filedescriptor)
{
    return syscall(SYS_close, filedescriptor);
}
```

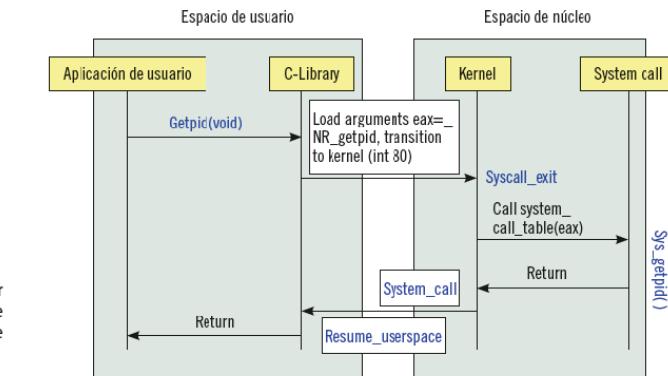
Esta forma se denomina directa porque se usa la llamada *syscall* proporcionada por el sistema operativo de forma nativa sin uso de API. En el primer parámetro, se especifica la llamada al sistema que se quiere ejecutar y el número y tipo del resto de parámetros dependerán de esta llamada.

Ejemplo de llamada al sistema usando la API *POSIX*:

```
#include <unistd.h>
#include <stdio.h>
#include <stdlib.h>
int main()
{
    pid_t rf;
    rf = fork();
}
```

Aquí se usa la función *fork()* de la API *POSIX*, la cual encapsula la llamada al sistema.

Diagrama de secuencia de una llamada a sistema a través de una función de la API *POSIX*



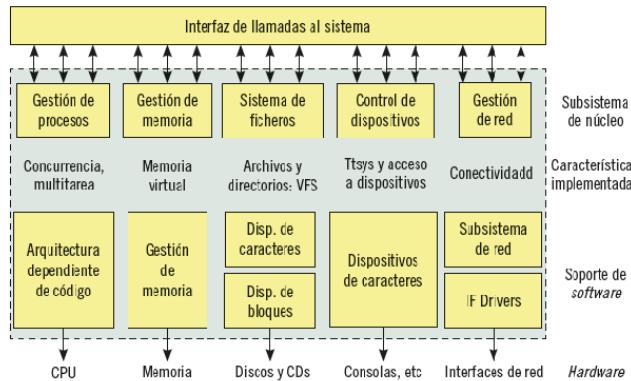
2.3. Principales tipos de llamadas al sistema

Existen cinco tipos de llamadas al sistema:

- Llamadas de control de procesos y comunicación entre procesos.
- Llamadas para la gestión de la memoria.
- Llamadas para el acceso a datos en ficheros.
- Llamadas para la gestión de dispositivos.
- Llamadas para la gestión de las comunicaciones por red.

A continuación, se van a explicar con más detalle estos grandes grupos de instrucciones que componen los subsistemas dentro del propio sistema operativo.

Diagrama de subsistema que componen la interfaz de llamadas al sistema



Control de procesos

Estas llamadas realizan tareas como crear/finalizar, bloquear/desbloquear, sincronizar, comunicar, depurar, obtener el identificador de un proceso, etc.

En los sistemas operativos primitivos, solo existía una aplicación que se ejecutaba y hasta que no finalizaba no se podía ejecutar otra. En los sistemas operativos moder-

nos, se ejecutan múltiples procesos a la vez. Incluso el concepto de proceso es independiente del de aplicación. Un programa puede estar compuesto de múltiples procesos.

El sistema operativo comienza un proceso cargándolo en la memoria y llamando a la función de entrada, que, en el caso de un sistema *Linux*, suele ser la función **main()**. Una de las llamadas del sistema básicas es **exit()**, que provoca la terminación del proceso. En *Unix* cada proceso posee un identificador, que se conoce con el nombre de PID (*Process ID*).



Recuerde

Un proceso puede entenderse como un programa en ejecución.

Existe una llamada al sistema muy importante en *Unix* que permite que un programa cree un nuevo proceso a partir del que se está ejecutando. Esta llamada al sistema es **fork()**. Sin embargo, debe existir una sincronización entre el proceso que se ejecuta y el nuevo proceso creado. La llamada al sistema **waitpid()** permite bloquear el proceso en ejecución hasta que el nuevo proceso creado sea finalizado. Por último, está la llamada al sistema **execve()**, que permite ejecutar un comando interno, comutando el proceso en ejecución por el nuevo proceso.

fork	Crea un nuevo proceso.
execve	Ejecuta un comando interno creando un nuevo proceso.
waitpid	Espera hasta que el proceso generado completa su ejecución.
exit	Termina la ejecución de un proceso.
getpid	Obtiene el identificador de proceso actual.
getppid	Obtiene el identificador del proceso padre del proceso actual.
nice	Cambia la prioridad de ejecución del proceso.

En *Windows*, una aplicación consiste en uno o más procesos. Un proceso, en términos simples, es un programa en ejecución. Cada proceso comienza su ejecución

con un solo hilo, denominado hilo primario, pero puede crear hilos adicionales desde cualquier hilo en ejecución. Un hilo es la unidad básica para la que el sistema operativo asigna tiempo de procesador y puede ejecutar cualquier parte del código de un proceso, incluyendo las partes que están siendo ejecutadas por otro hilo. Los hilos corren en el contexto de un proceso, por lo que comparten el espacio virtual y el sistema de recursos del propio proceso. Además, mantienen los siguientes sistemas:

- Manejadores de excepciones.
- Una prioridad de ejecución.
- Un conjunto de estructuras que el sistema usará para salvar el contexto de hilo hasta que sea despachado por el planificador.

El contexto de hilo incluye:

- El conjunto de registros de máquina del hilo.
- La pila del núcleo.
- Un bloque de entorno del hilo.
- Una pila de usuario en el espacio de direcciones del proceso del hilo.

CreateProcess es la llamada al sistema que permite crear un nuevo proceso y un hilo primario dentro de este. Este nuevo proceso se ejecuta dentro del contexto de seguridad del que realiza la llamada. *Windows* asigna un identificador que será válido hasta que finalice su ejecución. De esta forma, se puede identificar cualquier proceso del sistema, e incluso abrir un manejador al proceso usando la llamada al sistema **OpenProcess** y su identificador. Al hilo primario también se le asigna un identificador y de forma similar podrá usarse para crear un manejador a ese hilo a través de la llamada al sistema **OpenThread**. Por último, la llamada al sistema **ExitProcess** provoca la finalización del proceso y todos los hilos que lo contienen.

CreateProcess	Crea un proceso con un nuevo identificador.
OpenProcess	Devuelve un manejador del proceso con el identificador pasado a la llamada si existe.
GetCurrentProcess	Obtiene un manejador al proceso actual.
CreateThread	Crea un nuevo hilo dentro del proceso actual.
OpenThread	Devuelve un manejador del hilo con el identificador pasado a la llamada si existe.

CreateRemoteThread	Crea un hilo de ejecución en un proceso remoto.
GetCurrentThread	Obtiene un manejador al hilo actual.
ExitProcess	Termina la ejecución de un proceso.
WaitForInputIdle	El proceso se queda bloqueado hasta que finaliza la ejecución del proceso especificado en la llamada.



Actividades

3. ¿Cuál sería la secuencia de llamadas necesaria para crear un proceso y tras finalizar su ejecución crear un nuevo hilo?

Gestión de ficheros

Otra de las tareas principales del sistema operativo es la manipulación de ficheros. Existe un conjunto de llamadas al sistema que proporcionan los mecanismos adecuados para la manipulación de información en forma de ficheros.

En *Unix*, el sistema operativo asigna internamente a cada fichero abierto un descriptor o un identificador (usualmente un valor entero positivo). Cuando un proceso crea o abre un fichero, el sistema le devuelve este identificador para que se pueda manejar. Cada aplicación tiene sus propios descriptores de ficheros. Por convención, todo proceso, cuando se crea, posee tres descriptores de ficheros. El descriptor 0 identifica la entrada estándar (usada para recuperar datos del fichero), el 1 la salida estándar (usada para guardar datos en un fichero) y el 2 la salida estándar de errores (usada para registrar cada error que se produce). El resto de los descriptores son usados por ficheros ordinarios, tuberías, ficheros especiales o directorios.

Las principales llamadas al sistema para el manejo de los descriptores de ficheros son:

Open	Abre o crea un fichero, devolviendo el descriptor de ficheros al proceso que invoca la llamada.
Create	Crea un fichero, devolviendo el descriptor de ficheros al proceso que invoca la llamada.
Read	Permite leer un número de bytes de un fichero.
Write	Permite escribir un número de bytes en un fichero.
Close	Cierra un fichero, eliminando el descriptor de ficheros del proceso.
Lseek	Posiciona el cursor del fichero en una posición para que la llamada a read pueda leer la parte del fichero donde se ha posicionado.

Existen otras muchas llamadas al sistema básicas para el manejo de ficheros en *Unix*.

En *Windows*, la llamada al sistema **CreateFile** crea un nuevo fichero o abre uno existente. Para ello, hay que especificar el nombre del fichero, las instrucciones de creación y otros atributos. Cuando una aplicación crea un nuevo fichero, el sistema operativo añade este al directorio especificado.

Windows asigna un identificador único, llamado **manejador**, a cada fichero que es abierto o creado usando esta llamada al sistema. Una aplicación podrá usar ese manejador para realizar cualquier operación de lectura o escritura. Cuando una aplicación comienza, esta hereda todos los manejadores del proceso que ejecutó la aplicación.

Una aplicación debe cerrar todos los ficheros cuando no vaya a hacer uso de ellos usando la función **CloseHandle**. También se puede hacer uso de la llamada **DeleteFile** para borrar un fichero, aunque no será borrado hasta que todos los manejadores de este sean liberados.

Se podrá escribir y leer la información de un fichero a través de las llamadas **ReadFile**, **ReadFileEx**, **WriteFile** y **WriteFileEx**. Estas llamadas requieren el manejador del fichero abierto en modo lectura y/o escritura respectivamente. Estas llamadas especifican el número de bytes a leer y/o escribir a partir de la posición de localización que se indique. Para posicionar el cursor del fichero, también existe la llamada **SetFilePointer**.



Recuerde

En la terminología de *Windows*, un descriptor de ficheros es un manejador de ficheros.



Actividades

4. ¿Cuáles serían los descriptores de ficheros que existirían en un proceso donde se abren dos ficheros y se cierra uno?

Gestión de dispositivos

Los fabricantes de dispositivos de entrada/salida deben proporcionar la funcionalidad del dispositivo para la interfaz que el sistema operativo mantiene cuando necesita manejar dicho dispositivo. Esta interfaz es estándar en el caso de un sistema *Linux*:

open	Abre el dispositivo para su uso.
release	Cierra el dispositivo.
read	Lee un número de bytes del dispositivo.
write	Escribe un número de bytes en el dispositivo.
ioctl	Se utiliza para hacer un uso más personalizado del dispositivo.

Cuando el proveedor proporciona la funcionalidad para esta interfaz, se puede decir que el sistema es capaz de usar las capacidades del dispositivo.

En *Linux*, todos los dispositivos son tratados como ficheros. Por ello, se utilizan las llamadas al sistema específicas para el manejo de ficheros.

En *Windows*, las llamadas al sistema **ReadFile** y **WriteFile** también son utilizadas para la escritura y lectura de un dispositivo de entrada/salida. Pero existe una llamada

al sistema similar a **ioctl** para el control de los dispositivos: **DeviceControl**. Esta llamada ejecuta operaciones de entrada y salida o bien devuelve información sobre el dispositivo.

Información del sistema

Se ha visto cómo algunas llamadas al sistema permiten realizar operaciones sobre ficheros y sobre dispositivos. Sin embargo, existen otras que únicamente devuelven información para ser interpretada por la aplicación y realizar una acción u otra en función de las circunstancias. Estas llamadas al sistema son muy variadas y se encuentran en los cinco grupos principales. En *Linux*, algunas de estas llamadas son **fstat**, que permite obtener información acerca de un fichero específico (permisos, ruta absoluta, último tiempo de modificación, etc.); **lstat**, que realiza la misma función que **fstat** pero sobre enlaces simbólicos; **uname**, que devuelve información sobre el *kernel* (nombre del sistema, versión, nombre del dominio, etc.), y **gethostname**, que obtiene el nombre del *host*.

En *Windows*, algunas de estas llamadas son **GetCurrentProcessorNumber**, que devuelve el número del procesador del hilo actual.

Comunicaciones

Si se habla de comunicaciones entre procesos, en *Linux* hay varios mecanismos que lo permiten:

▪ **Semáforos.** Este mecanismo usa estructuras de datos y llamadas al sistema que permiten sincronizar dos o más procesos, permitiendo la comunicación entre ellos si se usan adecuadamente. Algunas de las llamadas al sistema que permiten el uso de estos mecanismos son: **semget** (proporciona un identificador de semáforo), **semop** (controla las operaciones que se realizan sobre el semáforo) y **semctl**.

▪ **Tuberías.** Son el mecanismo de comunicación entre procesos más primitivo de *Linux*. Proporcionan un flujo unidireccional de comunicación entre los procesos del mismo sistema. Para usar este mecanismo, se utiliza la llamada al sistema **pipe** (crea una tubería). Esta llamada crea dos descriptoros de ficheros que se usan en la comunicación. *Windows* también permite esta técnica de comunicación. Algunas de las llamadas al sistema serían: **CreateFile** (para crear un tubería anónima), **CreateNamedPipe** (para crear una tubería con nombre), **ReadFile** (leer a través de la tubería) y **WriteFile** (escritura en la tubería).

▪ **Sockets.** Son el sistema más popular de comunicación entre procesos. Esto es debido a que este mecanismo permite la comunicación entre los procesos que se encuentran en diferentes máquinas. Generalmente, son asociados con el concepto de comunicación de red en una arquitectura cliente/servidor, aunque

las posibilidades son mucho mayores. Las diferentes llamadas al sistema que se pueden encontrar para el uso de este mecanismo son: **socket** (devuelve un descriptor de socket para la comunicación), **bind** (enlaza un socket a un nombre e inicializa la estructura adecuada para su control), **listen** (mantiene el socket a la escucha de conexiones), **accept** (acepta una conexión sobre un socket), y **connect** (realiza una conexión sobre un socket).

▪ **Memoria compartida.** Se basa, como su propio nombre indica, en establecer una zona de la memoria que sea común y compartida para los procesos que intervienen en la comunicación. Las llamadas al sistema que gestionan este mecanismo son: **shmget** (obtiene el espacio de memoria compartida), **shmat** (enlaza la memoria compartida), **shmdt** (libera la memoria compartida) y **shmctl** (permite control el espacio de memoria compartida).

Algunas de las system calls más importantes que se encargan del manejo de las comunicaciones en Linux

semget	Crea un semáforo y devuelve su identificador.
semop	Realiza operaciones sobre el semáforo de forma atómica.
semctl	Realiza operaciones de control sobre el semáforo.
pipe	Crea una tubería: dos descriptoros para la lectura y escritura.
socket	Crea un socket y devuelve su identificador.
bind	Enlaza un socket a una estructura que define sus características.
listen	Escucha peticiones de conexión a través de un socket.
accept	Acepta una conexión por un socket.
connect	Conecta un socket a otro para establecer una comunicación.
shmget	Devuelve un identificador a un segmento de memoria compartida.

shmat	Enlaza un segmento de memoria compartida a un espacio de direcciones del proceso que realiza la llamada.
shmdt	Desenlaza un segmento de memoria compartida del espacio de direcciones del proceso que realiza la llamada.
shmctl	Ejecuta operaciones de control sobre la memoria compartida.

Windows mantiene algunas de estas técnicas, pero también implementa otras muchas, como por ejemplo:

▪ **RPC:** permite a las aplicaciones realizar llamadas remotas. RPC puede establecer comunicaciones entre procesos de un mismo sistema y procesos de sistemas conectados por red.

▪ **COM:** se trata del modelo de componentes exclusivo de Microsoft que permite la creación de objetos OLE para la interoperabilidad de las aplicaciones con otros programas escritos en diferentes lenguajes.



Actividades

5. ¿Cuál es la principal ventaja del mecanismo de comunicación por socket con respecto a otro mecanismo cualquiera?

2.4. Descripción y uso de las API estándar de uso común para llamadas a sistema

Como se ha visto en apartados anteriores, el uso directo de las llamadas al sistema no es usado frecuentemente, ya que requiere conocimientos muy profundos acerca del sistema. Las API surgen para resolver este gran problema y hacer más productivo el desarrollo de aplicaciones. A continuación, se detallan los API que son referencias actuales para los programadores.

Win32 API (sistemas Windows)

Las aplicaciones Windows se caracterizan por ser interactivas, lo cual implica que el diseño de una aplicación Windows está basado en eventos. Por lo tanto, el desarrollo de un programa Windows está fuertemente acoplado a la interfaz gráfica que permite su uso.

API Win32 es un conjunto muy extenso de funciones, tipos y mensajes para poder programar y desarrollar aplicaciones sobre sistemas Windows. Su nombre proviene de la necesidad que hubo en extender la API de los sistemas operativos de Windows de 16 bits a la nueva arquitectura de 32 bits. Está desarrollada en lenguaje C.

La mayoría de las funciones de esta API se agrupan en bibliotecas que adoptan forma de ficheros. Por lo general, con terminación ".dll" (*dynamic linked library*).

La API Win32 encapsula todas las llamadas al sistema en un sistema operativo Windows y proporciona una interfaz común para el desarrollo de este tipo de aplicaciones.

Un ejemplo de aplicación en Win32 es el siguiente:

```
#include <windows.h>

const char g_szClassName[] = "myWindowClass";

// Step 4: the Window Procedure
LRESULT CALLBACK WndProc(HWND hwnd, UINT msg, WPARAM wParam, LPARAM lParam)
```

```
{
    switch(msg)
    {
        case WM_CLOSE:
            DestroyWindow(hwnd);
            break;
        case WM_DESTROY:
            PostQuitMessage(0);
            break;
        default:
            return DefWindowProc(hwnd, msg, wParam, lParam);
    }
    return 0;
}

int WINAPI WinMain(HINSTANCE hInstance, HINSTANCE hPrevInstance,
    LPSTR lpCmdLine, int nCmdShow)
{
    WNDCLASSEX wc;
    HWND hwnd;
    MSG Msg;
    //Step 1: Registering the Window Class
    wc.cbSize      = sizeof(WNDCLASSEX);
    wc.style       = 0;
    wc.lpfnWndProc = WndProc;
    wc.cbClsExtra  = 0;
    wc.cbWndExtra  = 0;
    wc.hInstance   = hInstance;
    wc.hIcon       = LoadIcon(NULL, IDI_APPLICATION);
    wc.hCursor     = LoadCursor(NULL, IDC_ARROW);
```

```
    wc.hbrBackground = (HBRUSH)(COLOR_WINDOW+1);
    wc.lpszMenuName = NULL;
    wc.lpszClassName = g_szClassName;
    wc.hIconSm     = LoadIcon(NULL, IDI_APPLICATION);

    if(!RegisterClassEx(&wc))
    {
        MessageBox(NULL, "Window Registration Failed!", "Error!",
            MB_ICONEXCLAMATION | MB_OK);
        return 0;
    }

    // Step 2: Creating the Window
    hwnd = CreateWindowEx(
        WS_EX_CLIENTEDGE,
        g_szClassName,
        "The title of my window",
        WS_OVERLAPPEDWINDOW,
        CW_USEDEFAULT, CW_USEDEFAULT, 240, 120,
        NULL, NULL, hInstance, NULL);
    if(hwnd == NULL)
    {
        MessageBox(NULL, "Window Creation Failed!", "Error!",
            MB_ICONEXCLAMATION | MB_OK);
        return 0;
    }

    ShowWindow(hwnd, nCmdShow);
    UpdateWindow(hwnd);
```

```
// Step 3: The Message Loop
while( GetMessage(&Msg, NULL, 0, 0) > 0 )
{
    TranslateMessage(&Msg);
    DispatchMessage(&Msg);
}
return Msg.wParam;
}
```

Generalmente, este es el programa más simple que se puede escribir en un sistema *Windows*. Su ejecución muestra una ventana simple sin diálogo ni menús.

En el código se pueden observar múltiples llamadas a la API *Win32* que encapsulan llamadas al sistema operativo: **CreateWindowEx**, **GetMessage**, **ShowWindow**, **DispatchMessage**, etc.

[POSIX API \(sistemas Unix, Linux, macOS\)](#)

POSIX (Portable Operating System Interface) es un estándar de interfaz de sistemas operativos portables basado en el sistema operativo *Unix*. Esta es, sin duda, la mejor definición que se puede dar de *POSIX*. Surge como necesidad de buscar una interfaz que sea portable entre las distintas implementaciones de *Unix*.

Fue desarrollado dentro de la IEEE con la referencia 1003 y también se ha desarrollado como estándar internacional con la referencia ISO/IEC 9945. *POSIX* se encuentra en continua evolución y está compuesto por una familia de estándares que cubren diferentes aspectos del funcionamiento de un sistema operativo *Unix*.

Todos estos estándares se pueden agrupar en tres categorías: estándares de base (definen interfaces relacionadas con el sistema operativo), estándares en diferentes lenguajes de programación (realizan una traducción de un estándar de base a un lenguaje concreto, con el fin de que se pueda utilizar en ese lenguaje); y los entornos de sistemas abiertos (se componen de las funciones necesarias y los parámetros necesarios para que se pueda desplegar un estándar *POSIX* en un sistema predefinido).

Las características más relevantes son:

- Algunos tipos de datos utilizados por las funciones no se definen como parte del estándar, pero se definen como parte de la implementación. Estos tipos se

encuentran definidos en el archivo de cabecera "<sys/types.h>". Estos tipos acaban con el sufijo "_t". Por ejemplo: **uid_t** es un tipo que se emplea para almacenar un identificador de usuario (UID).

- Los nombres de las llamadas al sistema en *POSIX* son en general cortos y con todas sus letras en minúsculas. Ejemplos: **fork**, **close**, **read**.
- Las funciones normalmente devuelven 0 si se ejecutaron con éxito, 0 o -1 en caso de error. Cuando una función devuelve -1, se almacena en una variable global, denominada **errno**, el código de error. Este código de error es un valor entero. La variable **errno** se encuentra definida en el archivo de cabecera "<errno.h>".
- La mayoría de los recursos gestionados por el sistema operativo se refieren mediante descriptor. Un descriptor es un número entero mayor o igual que cero. Ejemplo:

```
#include <sys/types.h>
#include <stdio.h>
main(){
pid_t id_proceso;
pid_t id_padre;

id_proceso = getpid();
id_padre = getppid();
printf("Identificador de proceso: %d\n", id_proceso);
printf("Identificador del proceso padre %d\n", id_padre);
}
```



Sabía que...

La familia de estándares *POSIX* especifica las interfaces de usuario y software del sistema operativo en 15 documentos distintos.

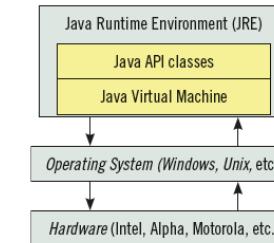
[Java API \(multiplataforma\)](#)

Java utiliza un formato especial de archivo que es interpretado por la *Máquina Virtual de Java (JVM)*, conocido como *bytecodes*. Este fichero contiene las instrucciones de un procesador ficticio que se creó para soportar *Java*. Sin embargo, nunca llegó a construirse un procesador similar de forma física y *JVM* realiza una emulación sobre ese tipo de procesador.

JVM interpreta los *bytecodes* de *Java* y los traduce dentro de acciones u operaciones de llamadas al sistema. Diferentes sistemas operativos realizan estas operaciones de diferentes maneras. Sin embargo, el programador de *Java* no necesita saber esos detalles. Es responsabilidad de la *JVM* manejar esas traducciones, por lo que el sistema operativo y la arquitectura CPU en la que *Java* está ejecutándose son completamente irrelevantes para el desarrollador.

En resumen, la *JVM* proporciona independencia en cuanto al sistema operativo a utilizar y a la arquitectura de la CPU.

Flujo de ejecución entre la máquina virtual de *Java* y el hardware del sistema



Propiamente dicho, *Java API* proporciona un conjunto de funciones y estructuras que permiten realizar una traducción a las llamadas al sistema operativo subyacente.



Actividades

- ¿Por qué se considera la API de *Java* como una API que envuelve llamadas al sistema?

3. Programas de utilidades y comandos del sistema

Todo sistema operativo posee herramientas que permiten la configuración de la mayoría de los aspectos importantes sobre el funcionamiento del mismo. En algunos, estas herramientas se proporcionan como aplicaciones de ventana (caso de Windows) que permiten una configuración más intuitiva y requieren menos conocimientos técnicos a nivel de sistema operativo para resolver tareas simples sobre el mantenimiento del sistema. Otras veces, son comandos del sistema que son ejecutados tras un terminal de texto y que, en la mayoría de los casos, requieren unos conocimientos más profundos, pero permiten un grado de configuración total.

La definición de comando es muy simple, se trata de un programa que se ejecuta y produce un resultado. Este programa puede ser ejecutado desde una línea de comandos en modo texto o bien puede ser el resultado de un evento producido por la interacción con un sistema de ventanas.

En cualquier caso, existen dos tipos de comandos: internos (comandos nativos del sistema y residentes en el núcleo del sistema operativo y, por lo tanto, en memoria) y externos, que residen en memoria masiva y de los que es necesario conocer exactamente la ubicación para poder ejecutarlos.

3.1. Principales tipos

Los sistemas Unix, como Linux, son sistemas operativos basados en la ejecución a través de línea de comandos y esto hace que sean sistemas con un número muy elevado de comandos internos. En concreto, se puede estar hablando de más de 400 comandos para realizar tareas de todo tipo.

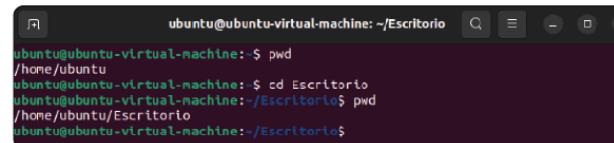
Windows, al ser un sistema operativo basado en interfaz gráfica, no requiere el uso de tanto comando interno por parte del administrador y, por lo tanto, no mantiene un listado tan amplio.

Operaciones con ficheros y directorios

En Linux, algunos de los comandos internos que se utilizan para el manejo de ficheros y directorios son los que se detallan a continuación.

pwd

Este comando permite mostrar el directorio actual donde se está en el sistema de ficheros.

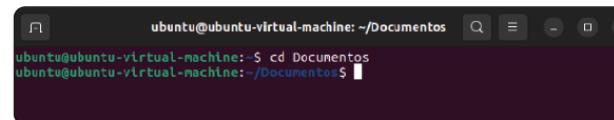


```
ubuntu@ubuntu-virtual-machine: ~/Escritorio
ubuntu@ubuntu-virtual-machine: $ pwd
/home/ubuntu
ubuntu@ubuntu-virtual-machine: $ cd Escritorio
ubuntu@ubuntu-virtual-machine:~/Escritorio$ pwd
/home/ubuntu/Escritorio
ubuntu@ubuntu-virtual-machine:~/Escritorio$
```

Pwd

cd

Se utiliza para cambiar de directorio de trabajo. El parámetro que se utiliza es la ubicación del nuevo directorio, que puede ser una ruta relativa o absoluta.



```
ubuntu@ubuntu-virtual-machine: ~/Documentos
ubuntu@ubuntu-virtual-machine: $ cd Documentos
ubuntu@ubuntu-virtual-machine:~/Documentos$
```

Cd

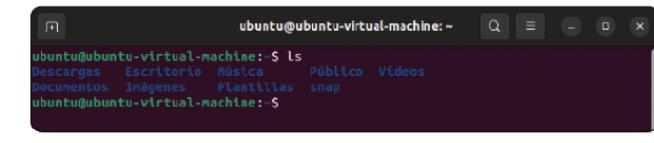


Sabía que...

cd es el acrónimo de *change directory*.

ls

Este comando muestra los ficheros y directorios que hay dentro del directorio especificado o en el directorio de trabajo si no se le especifica.

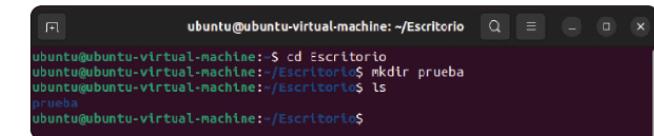


```
ubuntu@ubuntu-virtual-machine: ~
ubuntu@ubuntu-virtual-machine: $ ls
Descargas Escritorio Música Pública Videos
Documentos Ímágenes Plantillas snap
ubuntu@ubuntu-virtual-machine: $
```

Ls

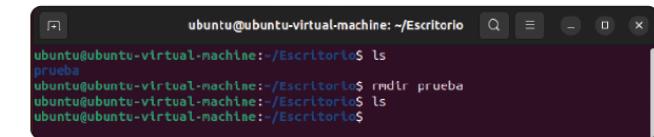
mkdir y rmdir

Se utilizan para crear y eliminar un directorio respectivamente. Siempre deben ir precedidos del nombre de un directorio.



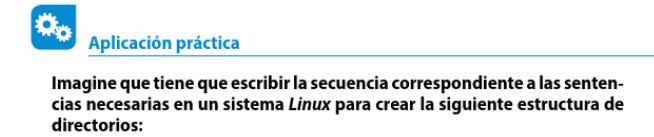
```
ubuntu@ubuntu-virtual-machine: ~/Escritorio
ubuntu@ubuntu-virtual-machine: $ cd Escritorio
ubuntu@ubuntu-virtual-machine:~/Escritorio$ mkdir prueba
ubuntu@ubuntu-virtual-machine:~/Escritorio$ ls
prueba
ubuntu@ubuntu-virtual-machine:~/Escritorio$
```

Mkdir



```
ubuntu@ubuntu-virtual-machine: ~/Escritorio
ubuntu@ubuntu-virtual-machine: $ ls
prueba
ubuntu@ubuntu-virtual-machine:~/Escritorio$ rmdir prueba
ubuntu@ubuntu-virtual-machine:~/Escritorio$ ls
ubuntu@ubuntu-virtual-machine:~/Escritorio$
```

Rmdir



```
ubuntu@ubuntu-virtual-machine: ~
ubuntu@ubuntu-virtual-machine: $ ls
prueba
ubuntu@ubuntu-virtual-machine:~/Escritorio$ rmdir prueba
ubuntu@ubuntu-virtual-machine:~/Escritorio$ ls
ubuntu@ubuntu-virtual-machine:~/Escritorio$
```



Aplicación práctica

Imagine que tiene que escribir la secuencia correspondiente a las sentencias necesarias en un sistema Linux para crear la siguiente estructura de directorios:

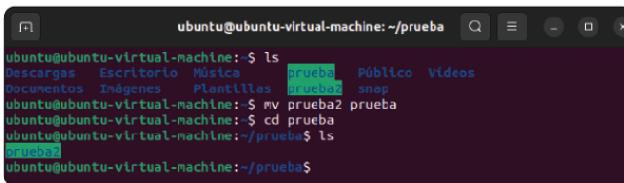
```
/prueba/primer/segunda  
/prueba/primer/tercera  
/prueba/cuarta  
¿Cómo lo haría?
```

SOLUCIÓN

```
# mkdir /prueba  
# mkdir /prueba/primer  
# mkdir /prueba/primer/segunda # mkdir /prueba/primer/tercera  
# mkdir /prueba/cuarta
```

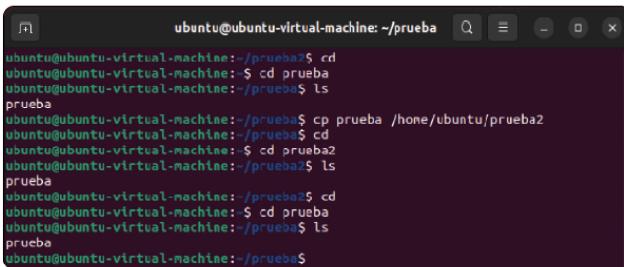
mv y cp

Se usan para mover y copiar ficheros y/o directorios respectivamente.



```
ubuntu@ubuntu-virtual-machine:~/prueba$ ls  
Descargas Escritorio Música prueba Público Videos  
Documentos Ímágenes Plantillas snap  
ubuntu@ubuntu-virtual-machine:~$ mv pruebaaz prueba  
ubuntu@ubuntu-virtual-machine:~$ cd prueba  
ubuntu@ubuntu-virtual-machine:~/prueba$ ls  
prueba  
ubuntu@ubuntu-virtual-machine:~/prueba$
```

Rmdir

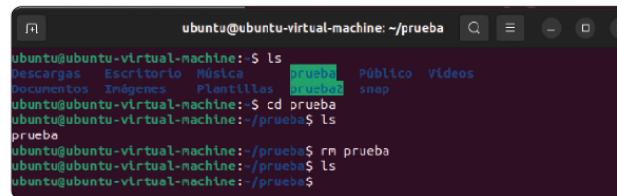


```
ubuntu@ubuntu-virtual-machine:~/prueba$ cd  
ubuntu@ubuntu-virtual-machine:~$ cd prueba  
ubuntu@ubuntu-virtual-machine:~/prueba$ ls  
prueba  
ubuntu@ubuntu-virtual-machine:~/prueba$ cp prueba /home/ubuntu/prueba2  
ubuntu@ubuntu-virtual-machine:~/prueba$ cd  
ubuntu@ubuntu-virtual-machine:~$ cd prueba2  
ubuntu@ubuntu-virtual-machine:~/prueba$ ls  
prueba  
ubuntu@ubuntu-virtual-machine:~/prueba$ cd  
ubuntu@ubuntu-virtual-machine:~$ cd prueba  
ubuntu@ubuntu-virtual-machine:~/prueba$ ls  
prueba  
ubuntu@ubuntu-virtual-machine:~/prueba$
```

Cp

rm

Se aplica sobre uno o varios directorios para eliminarlos del sistema.

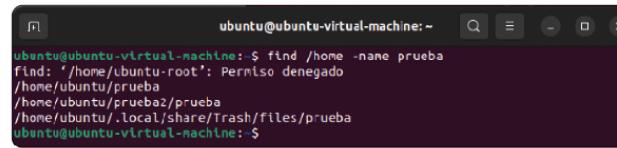


```
ubuntu@ubuntu-virtual-machine:~$ ls  
Descargas Escritorio Música prueba Público Videos  
Documentos Ímágenes Plantillas snap  
ubuntu@ubuntu-virtual-machine:~$ cd prueba  
ubuntu@ubuntu-virtual-machine:~/prueba$ ls  
prueba  
ubuntu@ubuntu-virtual-machine:~/prueba$ rm prueba  
ubuntu@ubuntu-virtual-machine:~/prueba$ ls  
ubuntu@ubuntu-virtual-machine:~/prueba$
```

Rm

find

Con este comando es posible buscar un fichero o directorio dentro del sistema de ficheros del sistema.



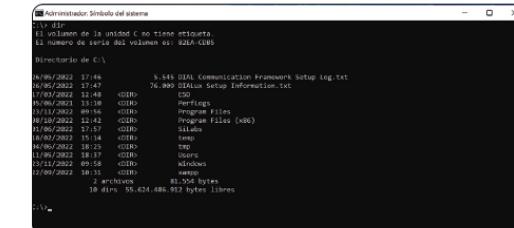
```
ubuntu@ubuntu-virtual-machine:~$ find /home -name prueba  
find: '/home/ubuntu-root': Permisos denegado  
/home/ubuntu/prueba  
/home/ubuntu/prueba/prueba  
/home/ubuntu/.local/share/trash/files/prueba  
ubuntu@ubuntu-virtual-machine:~$
```

Find

En Windows, se puede hacer uso de la siguiente lista de comandos para el manejo de ficheros y directorios:

Dir

El comando **dir** permite mostrar el contenido dentro de un directorio. Si no se indica un parámetro, mostrará el contenido del directorio actual.

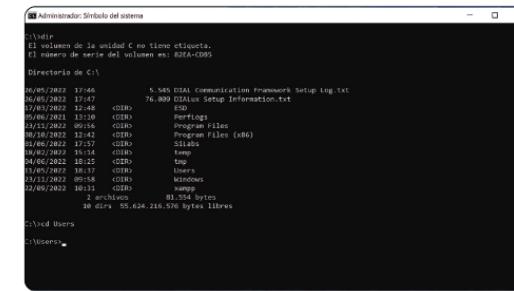


Administrator: Símbolo del sistema
C:\
El volumen de la unidad C no tiene etiqueta.
El número de serie del volumen es: 02EA-CB95
Directorio de C:\
16/05/2022 17:46 5,545 GMAIL Communication Framework Setup log.txt
16/05/2022 17:47 76,009 GIMP Setup Information.txt
17/05/2022 13:19 GDIR- PerfLogs
19/05/2022 09:56 GDIR- Program Files
20/05/2022 13:42 GDIR- Program Files (x86)
21/05/2022 17:57 GDIR- Sillas
24/05/2022 18:15 GDIR- Temp
11/05/2022 18:17 GDIR- Users
23/05/2022 18:17 GDIR- Windows
12/05/2022 10:31 GDIR-xampp
2 archives 81,554 bytes
16 dirs 55,624,216,576 bytes libres
C:\>

Dir

CD

Es un comando muy importante. Con él se puede cambiar el directorio de trabajo en el que se esté ubicado.



Administrator: Símbolo del sistema
C:\
El volumen de la unidad C no tiene etiqueta.
El número de serie del volumen es: 02EA-CB95
Directorio de C:\
16/05/2022 17:46 5,545 GMAIL Communication Framework Setup log.txt
16/05/2022 17:47 76,009 GIMP Setup Information.txt
17/05/2022 13:19 GDIR- PerfLogs
19/05/2022 09:56 GDIR- Program Files
20/05/2022 13:42 GDIR- Program Files (x86)
21/05/2022 17:57 GDIR- Sillas
24/05/2022 18:15 GDIR- Temp
11/05/2022 18:17 GDIR- Users
23/05/2022 18:17 GDIR- Windows
12/05/2022 10:31 GDIR-xampp
2 archives 81,554 bytes
16 dirs 55,624,216,576 bytes libres
C:\>cd Users
C:\Users\

CD

MKDIR

Crea un nuevo directorio. Si no se especifica una ubicación, el directorio nuevo se creará como subdirectorio del actual.

```
C:\Administrator: Símbolo del sistema

C:\Users\pruebas>dir
El volumen de la unidad C no tiene etiqueta.
El número de serie del volumen es: 82EA-CDB5

Directorio de C:\Users\pruebas

23/11/2022 12:28 <DIR> .
23/11/2022 12:28 <DIR> ..
0 archivos 0 bytes
2 dirs 55.608.340.480 bytes libres

C:\Users\pruebas>mkdir nuevo_directorio

C:\Users\pruebas>dir
El volumen de la unidad C no tiene etiqueta.
El número de serie del volumen es: 82EA-CDB5

Directorio de C:\Users\pruebas

23/11/2022 12:28 <DIR> .
23/11/2022 12:28 <DIR> ..
23/11/2022 12:28 <DIR> nuevo_directorio
0 archivos 0 bytes
3 dirs 55.608.274.944 bytes libres

C:\Users\pruebas>
```

MKDIR

RMDIR

Elimina un directorio vacío. No permite eliminar directorios con contenido. Únicamente se debe especificar el directorio que se quiere eliminar como parámetro del comando.

```
C:\Administrator: Símbolo del sistema

C:\Users\pruebas>dir
El volumen de la unidad C no tiene etiqueta.
El número de serie del volumen es: 82EA-CDB5

Directorio de C:\Users\pruebas

23/11/2022 12:28 <DIR> .
23/11/2022 12:28 <DIR> ..
23/11/2022 12:28 <DIR> nuevo_directorio
0 archivos 0 bytes
3 dirs 55.608.320.000 bytes libres

C:\Users\pruebas>rmdir nuevo_directorio

C:\Users\pruebas>dir
El volumen de la unidad C no tiene etiqueta.
El número de serie del volumen es: 82EA-CDB5

Directorio de C:\Users\pruebas

23/11/2022 12:29 <DIR> .
23/11/2022 12:28 <DIR> ..
0 archivos 0 bytes
2 dirs 55.608.320.000 bytes libres

C:\Users\pruebas>
```

RMDIR

TREE

Visualiza la estructura en árbol del subdirectorio indicado.

```
C:\Administrator: Símbolo del sistema

C:\Users>tree pruebas
Listado de rutas de carpetas
El número de serie del volumen es 82EA-CDB5
C:\USERS\PRUEBAS
    ├── directorio_1
    │   ├── directorio_1_A
    │   ├── directorio_1_B
    │   ├── directorio_1_C
    │   └── directorio_1_D
    └── directorio_2
        ├── directorio_2_1
        ├── directorio_2_2
        ├── directorio_2_3
        ├── directorio_2_4
        ├── directorio_2_5
        ├── directorio_2_6
        └── directorio_2_7

C:\Users>
```

TREE

COPY

Este comando se utiliza para copiar ficheros. Se puede cambiar el nombre del fichero de destino en la misma operación. Se deben especificar dos parámetros: el origen, que especifica la unidad, el directorio y/o fichero o ficheros que debe copiarse; y el destino, especificando la unidad, el directorio y el nombre donde se copiarán los ficheros.

```
C:\ Administrador: Símbolo del sistema

C:\Users\pruebas>dir
El volumen de la unidad C no tiene etiqueta.
El número de serie del volumen es: 82EA-CDB5

Directorio de C:\Users\pruebas

23/11/2022 12:38    <DIR>        .
23/11/2022 12:28    <DIR>        ..
23/11/2022 12:35           18 archivo.txt
23/11/2022 12:40    <DIR>        directorio_1
23/11/2022 12:32    <DIR>        directorio_2
      1 archivos       18 bytes
      4 dirs  55.603.806.208 bytes libres

C:\Users\pruebas>copy archivo.txt directorio_1
      1 archivo(s) copiado(s).

C:\Users\pruebas>cd directorio_1
C:\Users\pruebas\directorio_1>dir
El volumen de la unidad C no tiene etiqueta.
El número de serie del volumen es: 82EA-CDB5

Directorio de C:\Users\pruebas\directorio_1

23/11/2022 12:41    <DIR>        .
23/11/2022 12:38    <DIR>        ..
23/11/2022 12:35           18 archivo.txt
23/11/2022 12:30    <DIR>        directorio_1_A
23/11/2022 12:31    <DIR>        directorio_1_B
23/11/2022 12:31    <DIR>        directorio_1_C
      1 archivos       18 bytes
      5 dirs  55.603.806.208 bytes libres

C:\Users\pruebas\directorio_1>
```

COPY



Sabía que...

Si se escribe la palabra *help* antes de cualquier comando, se puede obtener una descripción de las opciones de ejecución que admite.

MOVE

Este comando permite mover o renombrar un fichero o directorio. Admite dos parámetros: el origen, que especifica la unidad, los ficheros y/o directorios a mover; y el destino, que especifica la unidad y la ubicación de los ficheros y/o directorios donde se van a mover.

```
C:\ Administrador: Símbolo del sistema

C:\Users\pruebas>dir
El volumen de la unidad C no tiene etiqueta.
El número de serie del volumen es: 82EA-CDB5

Directorio de C:\Users\pruebas

23/11/2022 12:45    <DIR>        .
23/11/2022 12:28    <DIR>        ..
23/11/2022 12:35           18 archivo.txt
23/11/2022 12:43    <DIR>        directorio_1
23/11/2022 12:32    <DIR>        directorio_2
      1 archivos       18 bytes
      4 dirs  55.603.179.520 bytes libres

C:\Users\pruebas>move archivo.txt directorio_1
Se han movido      1 archivos.

C:\Users\pruebas>dir
El volumen de la unidad C no tiene etiqueta.
El número de serie del volumen es: 82EA-CDB5

Directorio de C:\Users\pruebas

23/11/2022 12:45    <DIR>        .
23/11/2022 12:28    <DIR>        ..
23/11/2022 12:45    <DIR>        directorio_1
23/11/2022 12:32    <DIR>        directorio_2
      0 archivos       0 bytes
      4 dirs  55.603.179.520 bytes libres

C:\Users\pruebas>cd directorio_1
C:\Users\pruebas\directorio_1>dir
El volumen de la unidad C no tiene etiqueta.
El número de serie del volumen es: 82EA-CDB5

Directorio de C:\Users\pruebas\directorio_1

23/11/2022 12:45    <DIR>        .
23/11/2022 12:45    <DIR>        ..
23/11/2022 12:35           18 archivo.txt
23/11/2022 12:30    <DIR>        directorio_1_A
23/11/2022 12:31    <DIR>        directorio_1_B
23/11/2022 12:31    <DIR>        directorio_1_C
      1 archivos       18 bytes
      5 dirs  55.603.179.520 bytes libres

C:\Users\pruebas\directorio_1>
```

MOVE

MOVE

Este comando elimina ficheros. Se puede utilizar con uno o varios ficheros y utilizar comodines. No permite eliminar directorios.

```
C:\ Administrador: Símbolo del sistema

C:\Users\pruebas>dir
El volumen de la unidad C no tiene etiqueta.
El número de serie del volumen es: 82EA-CDB5

Directorio de C:\Users\pruebas\directorio_1

23/11/2022 12:45    <DIR>        .
23/11/2022 12:45    <DIR>        ..
23/11/2022 12:35           18 archivo.txt
23/11/2022 12:30    <DIR>        directorio_1_A
23/11/2022 12:31    <DIR>        directorio_1_B
23/11/2022 12:31    <DIR>        directorio_1_C
      1 archivos       18 bytes
      5 dirs  55.603.113.984 bytes libres

C:\Users\pruebas>del archivo.txt
C:\Users\pruebas>dir
El volumen de la unidad C no tiene etiqueta.
El número de serie del volumen es: 82EA-CDB5

Directorio de C:\Users\pruebas\directorio_1

23/11/2022 12:46    <DIR>        .
23/11/2022 12:45    <DIR>        ..
23/11/2022 12:30    <DIR>        directorio_1_A
23/11/2022 12:31    <DIR>        directorio_1_B
23/11/2022 12:31    <DIR>        directorio_1_C
      0 archivos       0 bytes
      5 dirs  55.603.015.680 bytes libres

C:\Users\pruebas\directorio_1>
```

DEL



Actividades

7. ¿Qué se quiere hacer si se escribe la siguiente secuencia de sentencias en una línea de comandos de un sistema operativo de Linux?

```
I # mkdir /prueba/
I # cd /prueba
I # cp /bin/ftp /prueba
```

Funciones de estado

En general, un sistema operativo debe contener comandos que permitan monitorizar el estado en el que se encuentra en cada momento.

```
C:\ Administrador: Símbolo del sistema

C:\Users\pruebas>dir
El volumen de la unidad C no tiene etiqueta.
El número de serie del volumen es: 82EA-CDB5

Directorio de C:\Users\pruebas

23/11/2022 12:38    <DIR>        .
23/11/2022 12:28    <DIR>        ..
23/11/2022 12:35           18 archivo.txt
23/11/2022 12:40    <DIR>        directorio_1
23/11/2022 12:32    <DIR>        directorio_2
      1 archivos       18 bytes
      4 dirs  55.603.806.208 bytes libres

C:\Users\pruebas>copy archivo.txt directorio_1
      1 archivo(s) copiado(s).

C:\Users\pruebas>cd directorio_1
C:\Users\pruebas\directorio_1>dir
El volumen de la unidad C no tiene etiqueta.
El número de serie del volumen es: 82EA-CDB5

Directorio de C:\Users\pruebas\directorio_1

23/11/2022 12:41    <DIR>        .
23/11/2022 12:38    <DIR>        ..
23/11/2022 12:35           18 archivo.txt
23/11/2022 12:30    <DIR>        directorio_1_A
23/11/2022 12:31    <DIR>        directorio_1_B
23/11/2022 12:31    <DIR>        directorio_1_C
      1 archivos       18 bytes
      5 dirs  55.603.806.208 bytes libres

C:\Users\pruebas\directorio_1>
```

COPY



Sabía que...

Si se escribe la palabra *help* antes de cualquier comando, se puede obtener una descripción de las opciones de ejecución que admite.

MOVE

Este comando permite mover o renombrar un fichero o directorio. Admite dos parámetros: el origen, que especifica la unidad, los ficheros y/o directorios a mover; y el destino, que especifica la unidad y la ubicación de los ficheros y/o directorios donde se van a mover.

```
C:\ Administrador: Símbolo del sistema

C:\Users\pruebas>dir
El volumen de la unidad C no tiene etiqueta.
El número de serie del volumen es: 82EA-CDB5

Directorio de C:\Users\pruebas

23/11/2022 12:45    <DIR>        .
23/11/2022 12:28    <DIR>        ..
23/11/2022 12:35           18 archivo.txt
23/11/2022 12:43    <DIR>        directorio_1
23/11/2022 12:32    <DIR>        directorio_2
      1 archivos       18 bytes
      4 dirs  55.603.179.520 bytes libres

C:\Users\pruebas>move archivo.txt directorio_1
Se han movido      1 archivos.

C:\Users\pruebas>dir
El volumen de la unidad C no tiene etiqueta.
El número de serie del volumen es: 82EA-CDB5

Directorio de C:\Users\pruebas

23/11/2022 12:45    <DIR>        .
23/11/2022 12:28    <DIR>        ..
23/11/2022 12:45    <DIR>        directorio_1
23/11/2022 12:32    <DIR>        directorio_2
      0 archivos       0 bytes
      4 dirs  55.603.179.520 bytes libres

C:\Users\pruebas>cd directorio_1
C:\Users\pruebas\directorio_1>dir
El volumen de la unidad C no tiene etiqueta.
El número de serie del volumen es: 82EA-CDB5

Directorio de C:\Users\pruebas\directorio_1

23/11/2022 12:45    <DIR>        .
23/11/2022 12:45    <DIR>        ..
23/11/2022 12:35           18 archivo.txt
23/11/2022 12:30    <DIR>        directorio_1_A
23/11/2022 12:31    <DIR>        directorio_1_B
23/11/2022 12:31    <DIR>        directorio_1_C
      1 archivos       18 bytes
      5 dirs  55.603.179.520 bytes libres

C:\Users\pruebas\directorio_1>
```

MOVE

MOVE

Este comando elimina ficheros. Se puede utilizar con uno o varios ficheros y utilizar comodines. No permite eliminar directorios.

```
C:\ Administrador: Símbolo del sistema

C:\Users\pruebas>dir
El volumen de la unidad C no tiene etiqueta.
El número de serie del volumen es: 82EA-CDB5

Directorio de C:\Users\pruebas\directorio_1

23/11/2022 12:45    <DIR>        .
23/11/2022 12:45    <DIR>        ..
23/11/2022 12:35           18 archivo.txt
23/11/2022 12:30    <DIR>        directorio_1_A
23/11/2022 12:31    <DIR>        directorio_1_B
23/11/2022 12:31    <DIR>        directorio_1_C
      1 archivos       18 bytes
      5 dirs  55.603.113.984 bytes libres

C:\Users\pruebas>del archivo.txt
C:\Users\pruebas>dir
El volumen de la unidad C no tiene etiqueta.
El número de serie del volumen es: 82EA-CDB5

Directorio de C:\Users\pruebas\directorio_1

23/11/2022 12:46    <DIR>        .
23/11/2022 12:45    <DIR>        ..
23/11/2022 12:30    <DIR>        directorio_1_A
23/11/2022 12:31    <DIR>        directorio_1_B
23/11/2022 12:31    <DIR>        directorio_1_C
      0 archivos       0 bytes
      5 dirs  55.603.015.680 bytes libres

C:\Users\pruebas\directorio_1>
```

DEL



Actividades

7. ¿Qué se quiere hacer si se escribe la siguiente secuencia de sentencias en una línea de comandos de un sistema operativo de Linux?

```
I # mkdir /prueba/
I # cd /prueba
I # cp /bin/ftp /prueba
```

Funciones de estado

En general, un sistema operativo debe contener comandos que permitan monitorizar el estado en el que se encuentra en cada momento.

uptime

Este comando muestra el tiempo que el sistema ha estado funcionando sin que haya sido reiniciado. Puede servir como indicativo de la calidad o estabilidad del sistema.

```
ubuntu@ubuntu-virtual-machine:~$ uptime
12:48:31 up 2:50, 1 user, load average: 0,52, 0,24, 0,19
ubuntu@ubuntu-virtual-machine:~$
```

Uptime

free

Si se necesita conocer la cantidad de memoria libre y usada que tiene el sistema, se puede utilizar este comando, que muestra la cantidad de memoria física, de intercambio, caché y de búfer consumida por el *kernel*.

```
ubuntu@ubuntu-virtual-machine:~$ free
total        used        free      shared  buff/cache   available
Memoria:    3982156   1820864   724412    17232   1436940   1861192
Swap:       2191356   133828   2057528
ubuntu@ubuntu-virtual-machine:~$
```

Free

vmstat

La información que muestra es sobre la memoria virtual, los procesos que se ejecutan, la cantidad de memoria de paginación, la cantidad de información manejada por los dispositivos de entrada/salida de bloques, la cantidad de cambios de contexto y la actividad de la CPU. Es, por lo tanto, un comando bastante completo y útil para el administrador.

La leyenda de los datos mostrados es la siguiente:

- r: número de procesos esperando su tiempo de ejecución.

■ b: número de procesos bloqueados a la espera de recursos.
■ w: número de procesos extraídos de la memoria de intercambio a la espera de ejecución.
■ swpd: memoria virtual empleada (KB).
■ free: memoria inactiva (KB).
■ buff: memoria empleada como búferes (KB).
■ cache: cantidad de memoria usada como caché (KB).
■ si: cantidad de memoria intercambiada desde el disco.
■ so: cantidad de memoria intercambiada al disco.
■ bi: bloques recibidos desde un dispositivo de bloques.
■ bo: bloques enviados a un dispositivo de bloques.
■ in: número de interrupciones por segundo, incluyendo el reloj.
■ cs: número de cambios de contexto entre procesos por segundo.
■ us: tiempo de ejecución en modo usuario.
■ sy: tiempo de ejecución en modo kernel.
■ id: tiempo sin ejecución.
■ wa: tiempo en espera por E/S.
■ st: tiempo robado desde una máquina virtual.

```
ubuntu@ubuntu-virtual-machine:~$ vmstat
procs .....memoria.....swap.....io.....sistema.....cpu.....
r b swpd libre búfer caché st so bl bo ln cs us sy id wa st
0 0 133828 749616 149992 1287296 1 9 126 149 146 264 3 2 95 0 0
ubuntu@ubuntu-virtual-machine:~$
```

Vmstat

iostat

Sirve para monitorizar la actividad de los dispositivos, particiones y sistemas de red (NFS) del sistema. Pero, además, permite obtener una media en porcentajes del uso de la CPU en el sistema.

Los primeros datos que aparecen muestran valores de consumo de CPU:

- %user: porcentaje de uso de la CPU cuando ejecuta código a nivel de usuario.
- %nice: porcentaje de uso de la CPU con prioridad nice.
- %system: porcentaje de uso de la CPU para el sistema.
- %iowait: porcentaje de uso de la CPU cuando se ha enviado una salida a un dispositivo E/S.

■ %steal: porcentaje de uso de la CPU cuando se realiza una petición de recurso a otra CPU.

■ %idle: porcentaje de uso de la CPU cuando se espera la respuesta de un dispositivo E/S.

Para cada dispositivo o partición del sistema operativo mostrará una línea con los siguientes datos:

- tps: transferencia a disco por segundo.
- Blk_read/s: cantidad de bloques leídos por segundo.
- Blk_wrtn/s: cantidad de bloques escritos por segundo.
- Blk_read: bloques totales leídos.
- Blk_wrtn: bloques totales escritos.

Device	tps	kB_read/s	kB_wrtn/s	kB_dscd/s	kB_read	kB_wrtn	kB_dscd
/dev	0,00	0,00	0,00	0,00	73	0	0
loop0	0,00	0,00	0,00	0,00	21	0	0
loop1	0,01	0,05	0,00	0,00	488	0	0
loop10	0,10	0,33	0,00	0,00	9740	0	0
loop11	0,00	0,03	0,00	0,00	350	0	0
loop12	0,01	0,13	0,00	0,00	1351	0	0
loop13	0,11	3,96	0,00	0,00	45393	0	0
loop14	0,00	0,00	0,00	0,00	18	0	0
loop2	0,01	0,10	0,00	0,00	1099	0	0
loop3	0,12	1,49	0,00	0,00	12511	0	0
loop4	0,01	0,11	0,00	0,00	2176	0	0
loop5	0,08	0,28	0,00	0,00	2897	0	0
loop6	0,33	3,34	0,00	0,00	34944	0	0
loop7	0,61	1,95	0,00	0,00	20377	0	0
loop8	1,08	3,43	0,00	0,00	35844	0	0
loop9	0,09	0,20	0,00	0,00	2102	0	0
sd0	10,29	229,35	292,36	0,00	2400084	3057077	0
sr0	0,02	0,51	0,00	0,00	5344	0	0

Iostat

df

Con este comando se muestra información en forma de lista de cada partición del sistema y la cantidad de espacio disponible en cada una. También se muestra información sobre el nombre de la partición, el espacio total, los bloques asignados y los bloques disponibles.

uptime

Este comando muestra el tiempo que el sistema ha estado funcionando sin que haya sido reiniciado. Puede servir como indicativo de la calidad o estabilidad del sistema.

```
ubuntu@ubuntu-virtual-machine:~$ uptime
12:48:31 up 2:50, 1 user, load average: 0,52, 0,24, 0,19
ubuntu@ubuntu-virtual-machine:~$
```

Uptime

free

Si se necesita conocer la cantidad de memoria libre y usada que tiene el sistema, se puede utilizar este comando, que muestra la cantidad de memoria física, de intercambio, caché y de búfer consumida por el kernel.

```
ubuntu@ubuntu-virtual-machine:~$ free
total        used        free      shared  buff/cache   available
Memoria:   3982156   1820804    724412     17232   1436940   1861192
Swap:      2191356   138828   2057528
ubuntu@ubuntu-virtual-machine:~$
```

Free

vmstat

La información que muestra es sobre la memoria virtual, los procesos que se ejecutan, la cantidad de memoria de paginación, la cantidad de información manejada por los dispositivos de entrada/salida de bloques, la cantidad de cambios de contexto y la actividad de la CPU. Es, por lo tanto, un comando bastante completo y útil para el administrador.

La leyenda de los datos mostrados es la siguiente:

■ r: número de procesos esperando su tiempo de ejecución.

- b: número de procesos bloqueados a la espera de recursos.
- w: número de procesos extraídos de la memoria de intercambio a la espera de ejecución.
- swpd: memoria virtual empleada (KB).
- free: memoria inactiva (KB).
- buff: memoria empleada como búferes (KB).
- cache: cantidad de memoria usada como caché (KB).
- si: cantidad de memoria intercambiada desde el disco.
- so: cantidad de memoria intercambiada al disco.
- bi: bloques recibidos desde un dispositivo de bloques.
- bo: bloques enviados a un dispositivo de bloques.
- in: número de interrupciones por segundo, incluyendo el reloj.
- cs: número de cambios de contexto entre procesos por segundo.
- us: tiempo de ejecución en modo usuario.
- sy: tiempo de ejecución en modo kernel.
- id: tiempo sin ejecución.
- wa: tiempo en espera por E/S.
- st: tiempo robado desde una máquina virtual.

```
ubuntu@ubuntu-virtual-machine:~$ vmstat
procs .....memoria.....-swap--.----io--.----sistema--.----cpu-----
r b swpd libre búfer caché sl so bi bo ln cs us sy id wa st
0 0 133828 749616 149992 1287296 1 9 126 149 146 264 3 2 95 0 0
ubuntu@ubuntu-virtual-machine:~$
```

Vmstat

iostat

Sirve para monitorizar la actividad de los dispositivos, particiones y sistemas de red (NFS) del sistema. Pero, además, permite obtener una media en porcentajes del uso de la CPU en el sistema.

Los primeros datos que aparecen muestran valores de consumo de CPU:

- %user: porcentaje de uso de la CPU cuando ejecuta código a nivel de usuario.
- %nice: porcentaje de uso de la CPU con prioridad nice.
- %system: porcentaje de uso de la CPU para el sistema.
- %iowait: porcentaje de uso de la CPU cuando se ha enviado una salida a un dispositivo E/S.

■ %steal: porcentaje de uso de la CPU cuando se realiza una petición de recurso a otra CPU.

■ %idle: porcentaje de uso de la CPU cuando se espera la respuesta de un dispositivo E/S.

Para cada dispositivo o partición del sistema operativo mostrará una línea con los siguientes datos:

- tps: transferencia a disco por segundo.
- Blk_read/s: cantidad de bloques leídos por segundo.
- Blk_wrtn/s: cantidad de bloques escritos por segundo.
- Blk_read: bloques totales leídos.
- Blk_wrtn: bloques totales escritos.

```
ubuntu@ubuntu-virtual-machine:~$ iostat
Linux 5.19.0-23-generic (ubuntu-virtual-machine) 23/11/22 _x86_64_ (2 CPU)
avg-cpu:  %user   %nice   %system   %iowait   %steal   %idle
           2,39    0,13    2,09    0,09    0,00  95,14

Device      tps   kB_read/s   kB_wrtn/s   kB_dscd/s   kB_read   kB_wrtn   kB_dscd
fd0       0,00      0,01      0,00      0,00      73          0          0
loop0     0,00      0,00      0,00      0,00      21          0          0
loop1     0,01      0,05      0,00      0,00     488          0          0
loop10    0,10      0,93      0,00      0,00    9740          0          0
loop11    0,00      0,03      0,00      0,00     359          0          0
loop12    0,01      0,13      0,00      0,00    1351          0          0
loop13    0,11      3,96      0,00      0,00   41393          0          0
loop14    0,00      0,00      0,00      0,00      18          0          0
loop2     0,01      0,10      0,00      0,00    1089          0          0
loop3     0,12      0,00      0,00      0,00   42871          0          0
loop4     0,01      0,11      0,00      0,00    1171          0          0
loop5     0,08      0,00      0,00      0,00     289          0          0
loop6     0,23      3,34      0,00      0,00    34944          0          0
loop7     0,61      1,95      0,00      0,00   20377          0          0
loop8     1,08      3,41      0,00      0,00    35846          0          0
loop9     0,09      0,20      0,00      0,00    2102          0          0
sda      10,99    229,55    292,36      0,00   240084   3057077          0
sr0      0,02      0,51      0,00      0,00     5344          0          0
ubuntu@ubuntu-virtual-machine:~$
```

iostat

df

Con este comando se muestra información en forma de lista de cada partición del sistema y la cantidad de espacio disponible en cada una. También se muestra información sobre el nombre de la partición, el espacio total, los bloques asignados y los bloques disponibles.

uptime

Este comando muestra el tiempo que el sistema ha estado funcionando sin que haya sido reiniciado. Puede servir como indicativo de la calidad o estabilidad del sistema.

```
ubuntu@ubuntu-virtual-machine:~$ uptime
12:48:31 up 2:50, 1 user, load average: 0,52, 0,24, 0,19
ubuntu@ubuntu-virtual-machine:~$
```

Uptime

free

Si se necesita conocer la cantidad de memoria libre y usada que tiene el sistema, se puede utilizar este comando, que muestra la cantidad de memoria física, de intercambio, caché y de búfer consumida por el kernel.

```
ubuntu@ubuntu-virtual-machine:~$ free
total        used        free      shared  buff/cache   available
Memoria:   3982156   1820804    724412     17232   1436940   1861192
Swap:      2191356   138828   2057528
ubuntu@ubuntu-virtual-machine:~$
```

Free

vmstat

La información que muestra es sobre la memoria virtual, los procesos que se ejecutan, la cantidad de memoria de paginación, la cantidad de información manejada por los dispositivos de entrada/salida de bloques, la cantidad de cambios de contexto y la actividad de la CPU. Es, por lo tanto, un comando bastante completo y útil para el administrador.

La leyenda de los datos mostrados es la siguiente:

■ r: número de procesos esperando su tiempo de ejecución.

- b: número de procesos bloqueados a la espera de recursos.
- w: número de procesos extraídos de la memoria de intercambio a la espera de ejecución.
- swpd: memoria virtual empleada (KB).
- free: memoria inactiva (KB).
- buff: memoria empleada como búferes (KB).
- cache: cantidad de memoria usada como caché (KB).
- si: cantidad de memoria intercambiada desde el disco.
- so: cantidad de memoria intercambiada al disco.
- bi: bloques recibidos desde un dispositivo de bloques.
- bo: bloques enviados a un dispositivo de bloques.
- in: número de interrupciones por segundo, incluyendo el reloj.
- cs: número de cambios de contexto entre procesos por segundo.
- us: tiempo de ejecución en modo usuario.
- sy: tiempo de ejecución en modo kernel.
- id: tiempo sin ejecución.
- wa: tiempo en espera por E/S.
- st: tiempo robado desde una máquina virtual.

```
ubuntu@ubuntu-virtual-machine:~$ vmstat
procs .....memoria.....-swap--.----io--.----sistema--.----cpu-----
r b swpd libre búfer caché sl so bi bo ln cs us sy id wa st
0 0 133828 749616 149992 1287296 1 9 126 149 146 264 3 2 95 0 0
ubuntu@ubuntu-virtual-machine:~$
```

Vmstat

iostat

Sirve para monitorizar la actividad de los dispositivos, particiones y sistemas de red (NFS) del sistema. Pero, además, permite obtener una media en porcentajes del uso de la CPU en el sistema.

Los primeros datos que aparecen muestran valores de consumo de CPU:

- %user: porcentaje de uso de la CPU cuando ejecuta código a nivel de usuario.
- %nice: porcentaje de uso de la CPU con prioridad nice.
- %system: porcentaje de uso de la CPU para el sistema.
- %iowait: porcentaje de uso de la CPU cuando se ha enviado una salida a un dispositivo E/S.

■ %steal: porcentaje de uso de la CPU cuando se realiza una petición de recurso a otra CPU.

■ %idle: porcentaje de uso de la CPU cuando se espera la respuesta de un dispositivo E/S.

Para cada dispositivo o partición del sistema operativo mostrará una línea con los siguientes datos:

- tps: transferencia a disco por segundo.
- Blk_read/s: cantidad de bloques leídos por segundo.
- Blk_wrtn/s: cantidad de bloques escritos por segundo.
- Blk_read: bloques totales leídos.
- Blk_wrtn: bloques totales escritos.

```
ubuntu@ubuntu-virtual-machine:~$ iostat
Linux 5.19.0-23-generic (ubuntu-virtual-machine) 23/11/22 _x86_64_ (2 CPU)
avg-cpu:  %user   %nice   %system   %iowait   %steal   %idle
           2,39    0,13    2,09    0,09    0,00  95,14

Device      tps   kB_read/s   kB_wrtn/s   kB_dscd/s   kB_read   kB_wrtn   kB_dscd
fd0       0,00      0,01      0,00      0,00      73          0          0
loop0     0,00      0,00      0,00      0,00      21          0          0
loop1     0,01      0,05      0,00      0,00     488          0          0
loop10    0,10      0,93      0,00      0,00    9740          0          0
loop11    0,00      0,03      0,00      0,00     359          0          0
loop12    0,01      0,13      0,00      0,00    1351          0          0
loop13    0,11      3,96      0,00      0,00   41393          0          0
loop14    0,00      0,00      0,00      0,00      18          0          0
loop2     0,01      0,10      0,00      0,00    1089          0          0
loop3     0,12      0,00      0,00      0,00   42871          0          0
loop4     0,01      0,11      0,00      0,00    1171          0          0
loop5     0,08      0,00      0,00      0,00     289          0          0
loop6     0,23      3,34      0,00      0,00    34944          0          0
loop7     0,61      1,95      0,00      0,00   20377          0          0
loop8     1,08      3,41      0,00      0,00    35846          0          0
loop9     0,09      0,20      0,00      0,00    2102          0          0
sda      10,99    229,55    292,36      0,00   240084   3057077          0
sr0      0,02      0,51      0,00      0,00     5344          0          0
ubuntu@ubuntu-virtual-machine:~$
```

Iostat

df

Con este comando se muestra información en forma de lista de cada partición del sistema y la cantidad de espacio disponible en cada una. También se muestra información sobre el nombre de la partición, el espacio total, los bloques asignados y los bloques disponibles.

```
ubuntu@ubuntu-virtual-machine:~$ df
Filesystem      Usado Disponibles Usado% Montado en
udev            0B             0B   0% /dev
tmpfs           398668 398668  0% /run
/dev/sda3        19945896 12620848 64% /dev/sda3
tmpfs           199576     0 100% /dev/shm
tmpfs           5120      4  5116  1% /run/lock
/dev/sda2        524252 5364 518888 2% /boot/efi
tmpfs           393212 4730 393470 2% /run/user/1000
/dev/sr0         143072 143072     0 0% /media/ubuntu/CDROM
/dev/fd0          1424      9 1415  1% /media/floppy0
ubuntu@ubuntu-virtual-machine:~$
```

df

who

Muestra los nombres de los usuarios conectados al sistema actualmente, el terminal que usan, el tiempo de conexión y el nombre del host desde el que se conectan.

```
ubuntu@ubuntu-virtual-machine:~$ who
ubuntu    tty2          2022-11-23 09:58 (tty2)
ubuntu-root  tty3          2022-11-23 12:53 (tty3)
ubuntu@ubuntu-virtual-machine:~$
```

Who

Edición y manipulación de ficheros

Otros comandos útiles son los que permiten manipular y modificar atributos de los ficheros como los permisos, el usuario y el grupo al que pertenece un fichero, o poder editar el contenido de un fichero. Los siguientes comandos permiten estas y otras muchas tareas de administración de ficheros en Linux.

chown y chgrp

Estos comandos se aplican sobre un fichero o directorio, y permiten modificar el usuario y grupo dueño del mismo respectivamente.

```
ubuntu@ubuntu-virtual-machine:~/prueba$ ls -al
total 12
drwxrwxrwx  2 ubuntu  ubuntu  4096 nov 23 13:06 .
drwxr-xr-x  20 ubuntu  ubuntu  4096 nov 23 12:17 ..
-rw-rw-r--  1 ubuntu  ubuntu  13 nov 23 12:14 prueba
ubuntu@ubuntu-virtual-machine:~/prueba$ chown roberto prueba
ubuntu@ubuntu-virtual-machine:~/prueba$ ls -al
total 12
drwxrwxrwx  2 ubuntu  ubuntu  4096 nov 23 13:06 .
drwxr-xr-x  20 ubuntu  ubuntu  4096 nov 23 12:17 ..
-rw-rw-r--  1 roberto  ubuntu  13 nov 23 12:14 prueba
ubuntu@ubuntu-virtual-machine:~/prueba$
```

Chown y chgrp

Si se quisiera cambiar el dueño del fichero file1 a root, se usaría la sentencia:

```
chown root file1
```

De la misma forma, si se necesita cambiar el grupo del fichero file1 al grupo group2, se usaría la sentencia:

```
chgrp group2 file1
```

chmod

Permite modificar los permisos de un fichero o directorio. En Linux existen tres grupos de permisos y, dentro de esos grupos, tres tipos de permisos:

- El primer grupo contiene los permisos que pertenecen al dueño del fichero (u).
- El segundo grupo posee los permisos que pertenecen al grupo del fichero (g).
- Y, finalmente, el tercer grupo contiene los permisos para el resto de los usuarios (o).

En cada grupo anterior se pueden otorgar o denegar permisos de lectura, escritura o ejecución. El permiso de lectura (r) permite poder leer el fichero, pero no modificarlo, el permiso de escritura (w) permite su modificación y, por último, el permiso de ejecución (x) permite ejecutar el fichero cuando se trata de un programa del sistema.

```
chmod u+rwx nombre_fichero
```

Si se quisiera otorgar permisos de escritura, lectura y ejecución al dueño del fichero, se escribiría la siguiente sentencia:

```
chmod g+w,o-w nombre_fichero
```

Si por el contrario se quisiera otorgar permiso de escritura a todos los usuarios que pertenecen al mismo grupo que el dueño del fichero y denegar la escritura al resto de usuarios, se escribiría:

```
root@localhost:~/prueba$ ls -al
total 12
drwxrwxrwx  2 root  root  4096 Mar 15 09:45 .
drwxr-xr-x  20 root  root  4096 Mar 15 04:43 ..
-rw-rw-r--  1 ayato root  5 Mar 15 09:45 fichero
root@localhost:~/prueba$ chmod g+w,o-x fichero
root@localhost:~/prueba$ ls -al
total 12
drwxr-xr-x  2 root  root  4096 Mar 15 09:45 .
drwxr-xr-x  20 root  root  4096 Mar 15 04:43 ..
-rw-rw-r--  1 ayato root  5 Mar 15 09:45 fichero
root@localhost:~/prueba$
```

Ejemplo de modificación de permisos de un fichero en Linux



Aplicación práctica

Imagine que se quiere cambiar el usuario y grupo de algunos de los ficheros del sistema. En concreto, para el fichero \prueba\fichero1, se va

a necesitar cambiar el usuario al que pertenece por el usuario root (usuario administrador del sistema), y el grupo root (grupo al que pertenece el usuario administrador del sistema). Para el fichero \prueba\fichero2, el cambio de usuario sería miusuario y el grupo root. ¿Cómo los haría desde la línea de comandos de Linux?

SOLUCIÓN

```
# chown root \prueba\fichero1
# chgrp root \prueba\fichero1
# chown miusuario \prueba\fichero2
# chgrp root \prueba\fichero2
```

emacs y vi

Son dos procesadores de texto que permiten editar ficheros. Los dos son bastante potentes y eso implica que sean complicados de utilizar si no se conocen bien.

En Windows, algunos comandos útiles para la manipulación de los ficheros son los siguientes:

attrib

Este comando muestra y/o cambia los atributos de un fichero o directorio del sistema.

Los permisos que se pueden modificar son:

- **+y-:** modificadores para asignar o liberar atributos de un fichero o directorio.
- **R:** atributo de solo lectura.
- **A:** atributo de archivo de almacenamiento.
- **S:** atributo de archivo de sistema.
- **H:** atributo de archivo oculto.

```
C:\Users\pruebas>attrib pruebas.txt
A          C:\Users\pruebas\pruebas.txt

C:\Users\pruebas>attrib +R pruebas.txt
A   R          C:\Users\pruebas\pruebas.txt

C:\Users\pruebas>
```

Otorgar permiso de lectura a un fichero



Actividades

8. ¿Qué se está haciendo cuando se ejecuta la siguiente sentencia en un sistema operativo Windows: attrib +H C:\Usuarios\Mi espacio?

type

Este comando permite mostrar en la línea de comando el contenido del fichero en modo texto (ASCII).

```
C:\Administrator: Símbolo del sistema
C:\Users\pruebas>dir
El volumen de la unidad C no tiene etiqueta.
El número de serie del volumen es: 82EA-CDB5

Directorio de C:\Users\pruebas

23/11/2022 15:02    <DIR>      .
23/11/2022 12:28    <DIR>      ..
23/11/2022 12:46    <DIR>      directorio_1
23/11/2022 12:32    <DIR>      directorio_2
23/11/2022 15:05            71 pruebas.txt
                           1 archivos           71 bytes
                           4 dirs   55.599.312.896 bytes libres

C:\Users\pruebas>type pruebas.txt
Este es el contenido que
se encuentra dentro del
archivo pruebas.txt
C:\Users\pruebas>
```

Type

Soporte para lenguajes de programación (compiladores, enlazadores, ensambladores, intérpretes, etc.)

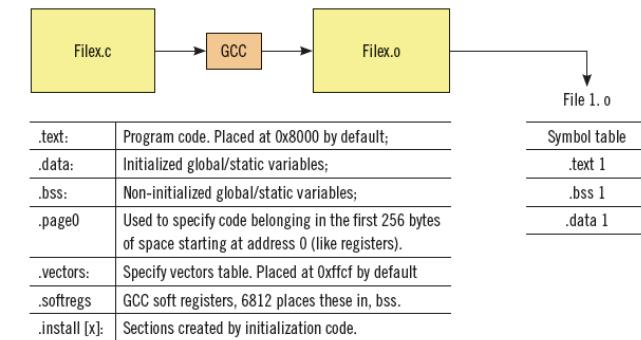
Linux es un sistema abierto al desarrollo de aplicaciones y a la continua mejora del sistema. Debido a esa filosofía, proporciona de forma nativa las herramientas básicas

necesarias para la programación. Entre estas herramientas, no pueden faltar compiladores, enlazadores y ensambladores. Los siguientes tienen una importancia vital en el esquema mantenido por la comunidad.

Compiladores

El compilador nativo por excelencia que está presente en casi todas las distribuciones de Linux es GCC (GNU Compiler Collection). En realidad, se puede ver como una familia de compiladores que permiten la compilación de lenguajes como C, C++, Objective C y Fortran. Este compilador es capaz de generar un ejecutable a partir de un fichero con código fuente en alguno de estos lenguajes.

Estructura lógica de un fichero objeto generado por el compilador GCC



AZ Definición

Compilación

Proceso de traducción de un código escrito en un lenguaje de programación a código en lenguaje máquina.

En Linux, se puede usar un editor de ficheros cualquiera para crear un programa basado en alguno de los lenguajes citados. Por ejemplo, un programa muy simple en lenguaje C sería el siguiente:

```
#include <stdio.h>

int main() {
    printf("hola mundo");
    return 0;
}
```

Si se guarda el contenido anterior en un fichero llamado "main.c", se podrá usar el compilador GCC para generar el ejecutable del programa de la siguiente manera:

```
ubuntu@ubuntu-virtual-machine:~/prueba$ ls
main.c  main.o  prueba
ubuntu@ubuntu-virtual-machine:~/prueba$ gcc -o main main.c
ubuntu@ubuntu-virtual-machine:~/prueba$ ls
main  main.c  main.o  prueba
ubuntu@ubuntu-virtual-machine:~/prueba$ ./main
hola mundo
ubuntu@ubuntu-virtual-machine:~/prueba$
```

Main.c



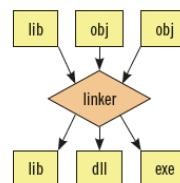
Actividades

9. ¿Por qué Linux proporciona de forma nativa el compilador GCC? Busque la explicación en internet.

Enlazadores

Al compilar un ejecutable hace falta enlazarlo con todas las bibliotecas que use el código que se compila. El programa o comando que permite enlazar todas esas bibliotecas se denomina **ld**, y es el enlazador nativo del proyecto GNU. Por lo general, el compilador GCC ya realiza una llamada a **ld** al compilar, por lo que no es necesario su uso de forma directa salvo en casos muy específicos.

Resultado de un enlazado



Ensambladores

Uno de los ensambladores más importantes de Linux es **GAS**, que es el ensamblador nativo del proyecto GNU. Permite generar un ejecutable a partir de código en ensamblador. Para ello, se debe codificar el programa en ensamblador en un fichero y realizar la llamada al comando **as** para compilarlo.

Supóngase que se tiene el siguiente trozo de código en un fichero "main.asm":

```
.text
.global _start
_start:
    movl $len, %edx
    movl $msg, %ecx
    movl $1, %ebx
    movl $4, %eax
    int $0x80
    movl $0, %ebx
    movl $1, %eax
    int $0x80

.data
msg: .ascii "hola mundo!\n"
len=12
```

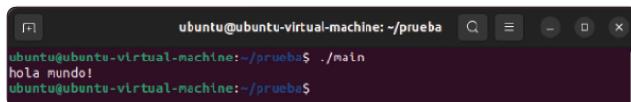
Este código está escrito en lenguaje ensamblador AT&T. Para compilarlo, se usa **as**, que generará un fichero ".o". Tras esto, se utiliza **ld** para enlazar y generar el ejecutable.

Para compilar el fichero "main.asm" que contiene código ensamblador, se ejecuta la sentencia: **as -o main.o main.asm**. Esto genera un fichero "main.o" con el código sin referencias resueltas. A continuación, se usa el enlazador para resolver esas referencias y obtener el fichero ejecutable. Se usa la sentencia: **ld -s -o main main.o**. El fichero ejecutable será **main**.

```
ubuntu@ubuntu-virtual-machine:~/prueba$ as -o main.o main.asm
ubuntu@ubuntu-virtual-machine:~/prueba$ ld -s -o main main.o
ubuntu@ubuntu-virtual-machine:~/prueba$
```

As-o main.o,ain.asm y ld-s-o main main.o

Finalmente, se podrá ejecutar el programa obtenido de la siguiente manera:



```
ubuntu@ubuntu-virtual-machine:~/prueba$ ./prueba
holo mundo!
ubuntu@ubuntu-virtual-machine:~/prueba$
```

Ejecutar el programa

En Windows no existe soporte nativo para el desarrollo de aplicaciones. Si se quiere comenzar a programar, se necesita instalar el compilador y enlazador adecuado. Sin embargo, esto no quiere decir que Microsoft no proporcione herramientas de desarrollo. Un ejemplo sería *Microsoft Visual Studio* que es una familia de compiladores, editores y enlazadores para el desarrollo de aplicaciones usando tecnologías como C#, J#, C++, Visual Basic, etc.

Intérpretes

Los intérpretes son programas que permiten la ejecución de instrucciones sin realizar ningún tipo de compilación previa a la ejecución del código. *Linux* viene acompañado de algunos intérpretes conocidos como *Perl* y *Python*.

Perl fue creado como complemento del sistema operativo y con la finalidad de simplificar las tareas de administración sobre el sistema. Además de ser *software libre*, es muy rápido y multiplataforma.

```
print " PENSANDO UN NUMERO ...\\n";
$numero = rand(100);
$entrada = $numero+1;
while ($entrada > $numero)
{
    print " Introduce un numero del 1 al 100, menor que el pensado:";
    $entrada = <STDIN>;
    if($entrada > $numero)
    {
        print "El numero introducido es mayor que el pensado.\\n";
    }
}

print " EL NUMERO PENSADO ERA:",$numero;
print "\\n";
```

Otro intérprete que acompaña a *Linux* es *Python*. Su principal característica es el uso de una sintaxis limpia, sin adornos, para favorecer la legibilidad del código. Al igual que *Perl*, se trata de un lenguaje libre y multiplataforma.

```
1. ###Imprimir los numeros impares desde el 1 al 25, ambos inclusive
2. n = 1
3. h = ''
4. while n <= 25:
5.     if n%2 != 0:
6.         h += ' %i' % n
7.     n += 1
8. print h
```

Otros

Otras herramientas que suele proporcionar el sistema operativo a los desarrolladores son los depuradores y editores de texto. Un depurador permite al programador realizar un seguimiento exhaustivo del programa que está desarrollando y detectar errores ejecutando el código paso a paso. Uno de los más importantes es el depurador *gdb* o depurador estándar del proyecto *GNU*. Es portable y funciona para programas escritos en C, C++ y Fortran.



Aplicación práctica

Está en un sistema *Linux* y quiere ejecutar una aplicación de la que tiene el código almacenado en el fichero "principal.asm". En concreto, se trata de código en ensamblador. Realice las llamadas a los comandos necesarios para compilar y ejecutar el código.

SOLUCIÓN

```
as -o principal.o principal.asm
ld -s -o principal principal.o
./principal
```

Los editores de texto son muy importantes porque permiten escribir el contenido de los programas. Actúan como entornos de desarrollo en la mayoría de los casos y algunos proporcionan funciones específicas dedicadas a la programación: resaltado de sintaxis, autoidentificación, etc.

Ejecución de programas

El principal objetivo de un sistema operativo es ofrecer los mecanismos necesarios para ejecutar las aplicaciones del usuario. Toda aplicación, es un proceso que se ejecuta en la unidad central de proceso. Es natural pensar que deben existir herramientas que permitan monitorizar la cantidad de procesos que, en cualquier momento se encontraban ejecutándose en el sistema. Para *Linux*, algunos de los comandos más utilizados se describen a continuación.

ps

Es uno de los comandos más importantes de *Linux*. Con él se pueden obtener casi todos los datos acerca de los programas que se están ejecutando en el sistema. Un ejemplo de ejecución sería: **ps aux**.

```
ubuntu@ubuntu-virtual-machine:~/prueba$ ps aux
USER      PID  %CPU %MEM   VSZ   RSS TTY      STAT START  TIME COMMAND
root      1  0.0  0.0 16959 12532 ?        0:00 /bin/sh /init a
root      2  0.0  0.0  0  0 ?        0:00 [kthreadd]
root      3  0.0  0.0  0  0 ?        0:00 [rcu_gp]
root      4  0.0  0.0  0  0 ?        1< 15:08 0:00 [rcu_par_gp]
root      5  0.0  0.0  0  0 ?        1< 15:08 0:00 [netns]
root      7  0.0  0.0  0  0 ?        1< 15:08 0:00 [kworker/0:0H]
root      9  0.0  0.0  0  0 ?        1< 15:08 0:00 [kworker/0:1H]
root     10  0.0  0.0  0  0 ?        1< 15:08 0:00 [mm_percpu_wq]
root     11  0.0  0.0  0  0 ?        I 15:08 0:00 [rcu_tasks_kt]
root     12  0.0  0.0  0  0 ?        I 15:08 0:00 [rcu_tasks_rj]
root     13  0.0  0.0  0  0 ?        I 15:08 0:00 [rcu_tasks_tr]
root     14  0.0  0.0  0  0 ?        S 15:08 0:00 [ksoftirqd/0]
root     15  0.0  0.0  0  0 ?        S 15:08 0:00 [ksoftirqd/1]
root     16  0.0  0.0  0  0 ?        S 15:08 0:00 [migration/0]
root     17  0.0  0.0  0  0 ?        S 15:08 0:00 [idle_inject/0]
root     19  0.0  0.0  0  0 ?        S 15:08 0:00 [cpuhp/0]
root    20  0.0  0.0  0  0 ?        S 15:08 0:00 [cpuhp/1]
```

ps

kill

El comando **kill** se utiliza para detener procesos que se están ejecutando en segundo plano. Para ello, se hace uso del identificador de proceso.

```
ubuntu 1984 0.0 0.6 344688 26424 ? Ssl 15:08 0:00 /usr/libexec/
ubuntu 1947 0.0 0.6 194312 25572 ? Sl 15:08 0:00 /usr/libexec/
root 2829 0.0 0.6 395696 33220 ? ssl 15:08 0:00 /usr/libexec/
ubuntu 2317 0.0 0.8 491184 32768 ? Sl 15:08 0:00 update-notif/
root 2429 0.0 0.0 171184 8784 ? Sl 15:10 0:00 /usr/libexec/
root 2437 0.1 0.6 0 0 ? I 15:13 0:01 [kworker/1:1]
ubuntu 2482 0.1 0.6 3236864 65192 ? Sl 15:14 0:01 gjs /usr/share/
ubuntu 2548 0.0 1.3 862784 54976 ? Sl 15:14 0:00 /usr/bin/gnom
ubuntu 2550 0.4 1.3 554584 53376 ? Rsl 15:14 0:04 /usr/libexec/
ubuntu 2665 0.0 0.1 11060 5260 pts/0 Ss 15:14 0:00 bash
ubuntu 2752 0.0 0.2 314532 8440 ? Sl 15:15 0:00 /usr/libexec/
root 2753 0.2 0.6 0 0 ? D 15:15 0:02 [kworker/u2:2]
root 2804 0.1 0.6 0 0 ? I 15:15 0:01 [kworker/u2:56]
root 3381 0.0 0.6 0 0 ? I 15:20 0:00 [kworker/0:5]
ubuntu 3397 0.5 3.5 1357116 142060 ? Sl 15:21 0:00 /usr/bin/naut
root 3446 0.1 0.6 0 0 ? I 15:22 0:00 [kworker/0:3]
root 3699 0.1 0.6 0 0 ? I 15:28 0:00 [kworker/1:2]
ubuntu 3747 0.0 0.6 12564 3544 pts/0 R+ 15:31 0:00 ps aux
```

Kill

top

Otro comando importante de *Linux* es **top**, el cual permite echar un vistazo a la actividad de la CPU en tiempo real, mostrando una lista con las tareas del sistema que están haciendo un uso más intenso de la CPU.

Para *Windows*, también existen este tipo de comandos, aunque debido a su naturaleza es más común el uso de la interfaz gráfica para la monitorización y manipulación de los procesos en ejecución.

```
ubuntu@ubuntu-virtual-machine:~/prueba$ top
top - 15:34:11 up 25 min, 1 user, load average: 0.88, 0.19, 0.25
Tasks: 291 total, 1 ejecutor, 290 kerner, 0 detener, 0 zombie
KCPUs(s): 0,3 us, 0,2 sy, 0,0 nl, 99,5 ld, 0,0 wa, 0,0 hi, 0,0 sl, 0,0 st
M1B Mem : 3888,8 total, 554,8 libre, 1850,2 usado, 1483,9 búfer/caché
M1B Intercambio: 2340,0 total, 2449,0 libre, 0,0 usado. 1756,9 dispon Mem

PID USUARIO PR NT VIRT RES SHR %CPU %MEM 100% ORDEN
1050 ubuntu 20 0 3896494 277728 133848 5 0,3 7,0 0:55:08 gnome-shell
1616 ubuntu 20 0 288304 38401 29756 5 0,3 1,0 0:03:45 ventoold
2755 root 20 0 0 0 0 0 I 0,3 0,0 0:02:62 kworker/0:2-events
3090 root 20 0 0 0 0 0 I 0,3 0,0 0:00:46 kworker/1:2-events
3758 root 20 0 0 0 0 I 0,3 0,0 0:00:12 kworker/u2:56:1-events_freeze+
3760 ubuntu 20 0 13304 4480 3448 R 0,3 0,1 0:00:05 top
1 root 20 0 167964 12532 8656 5 0,0 0,3 0:03:54 systrend
2 root 20 0 0 0 0 0 0,0 0,0 0:00:00 kthread
3 root 0:20 0 0 0 0 0 0,0 0,0 0:00:00 ksoftirqd
4 root 0:20 0 0 0 0 0 0,0 0,0 0:00:00 rcu_par_gp
5 root 0:20 0 0 0 0 0 0,0 0,0 0:00:00 netns
7 root 0:26 0 0 0 0 0 0,0 0,0 0:00:00 kworker/0:0H-events_highpri
9 root 0:26 0 0 0 0 0 0,0 0,0 0:00:27 kworker/0:1H-events_highpri
10 root 0:26 0 0 0 0 0 0,0 0,0 0:00:00 mm_percpu_wq
11 root 20 0 0 0 0 0 0,0 0,0 0:00:00 rcu_tasks_kthread
12 root 20 0 0 0 0 0 0,0 0,0 0:00:00 rcu_tasks_rude_kthread
```

Top

Tasklist y taskkill

Son dos comandos que permiten la manipulación de los procesos en ejecución desde la línea de comandos. **Tasklist** muestra el listado de procesos que están siendo ejecutados en el momento.

```
C:\Users>tasklist
Nombre de Imagen PID Nombre de sesión Num. de ses. uso de memoria
System Idle Process 0 Services 0 0 8 KB
System 4 Services 0 0 124 KB
svchost.exe 108 Services 0 7c 556 KB
smss.exe 336 Services 0 1 124 KB
rpsvc.exe 620 Services 0 5 703 KB
xinput.exe 604 Services 0 6 668 KB
audiosrv.exe 708 Services 0 9 408 KB
lascss.exe 748 Services 0 28 464 KB
svchost.exe 876 Services 0 32 512 KB
fontdrvhost.exe 908 Services 0 3 404 KB
cvchost.exe 1008 Services 0 16 412 KB
```

Tasklist

Taskkill permite detener un proceso que está siendo ejecutado y que se encuentra en la lista anterior. Para ello, se hace uso del **PID** del proceso.

```
C:\Users>taskkill /PID 7196
Correcto: se envio la señal de término al proceso con PID 7196.

C:\Users>
```

Taskkill

Actividades

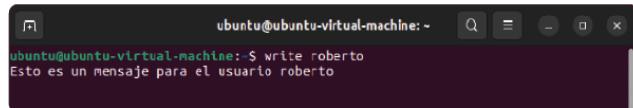
10. ¿Por qué son tan importantes comandos como ps o tasklist en los sistemas operativos modernos? Justifique su respuesta.

Comunicaciones, mensajería, intercambio remoto de archivos, etc.

Un sistema operativo que no pueda establecer comunicación entre sus usuarios no podría ser considerado como tal. Los SS. OO. actuales mantienen una infraestructura de red que en esencia ha evolucionado muy poco con el paso del tiempo. A continuación, se detallan algunos de los comandos que hacen uso de esa infraestructura y que mejoran la interacción entre la máquina y el usuario de forma notable.

Write

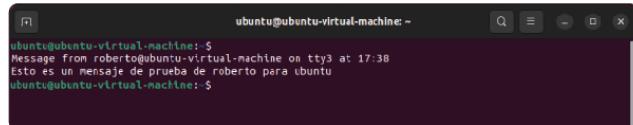
Este comando permite enviar mensajes entre usuarios que están conectados en ese momento al mismo sistema.



```
ubuntu@ubuntu-virtual-machine:~$ write roberto  
Esto es un mensaje para el usuario roberto
```

Write

El resultado es que aparece el mensaje en el terminal del usuario conectado:



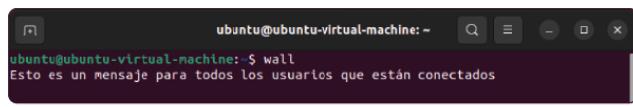
```
ubuntu@ubuntu-virtual-machine:~$  
Message from roberto@ubuntu-virtual-machine on pts/0 at 17:38  
Esto es un mensaje de prueba de roberto para ubuntu  
ubuntu@ubuntu-virtual-machine:~$
```

Mensaje en el terminal del usuario conectado

La comunicación es bidireccional, por lo que se puede mantener una conversación entre los dos usuarios.

Wall

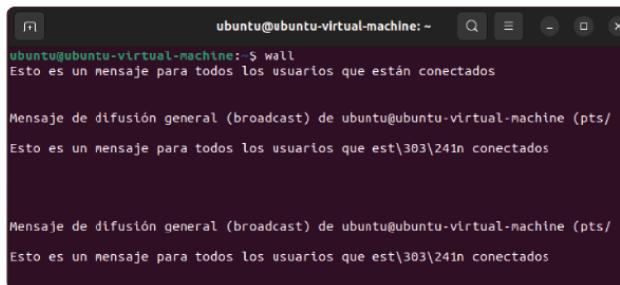
Se usa para mandar un mensaje a todos los usuarios conectados en ese momento al sistema. Se trata de una comunicación unidireccional.



```
ubuntu@ubuntu-virtual-machine:~$ wall  
Esto es un mensaje para todos los usuarios que están conectados
```

Wall

El resultado es enviado y mostrado en el terminal de cada usuario conectado:



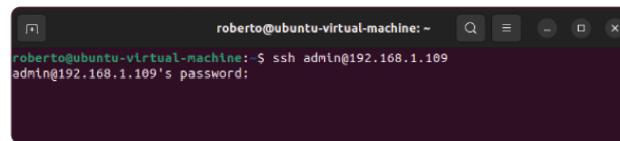
```
ubuntu@ubuntu-virtual-machine:~$ wall  
Esto es un mensaje para todos los usuarios que están conectados  
  
Mensaje de difusión general (broadcast) de ubuntu@ubuntu-virtual-machine (pts/0)  
Esto es un mensaje para todos los usuarios que están conectados  
  
Mensaje de difusión general (broadcast) de ubuntu@ubuntu-virtual-machine (pts/1)  
Esto es un mensaje para todos los usuarios que están conectados
```

Mensaje en el terminal del usuario conectado

ssh

SSH (*Secure Shell*) permite realizar una conexión a otra máquina a través de la red y ejecutar un comando en dicha máquina. Incluso se pueden mover ficheros entre ellas, y lo mejor de todo es que se trata de una comunicación codificada.

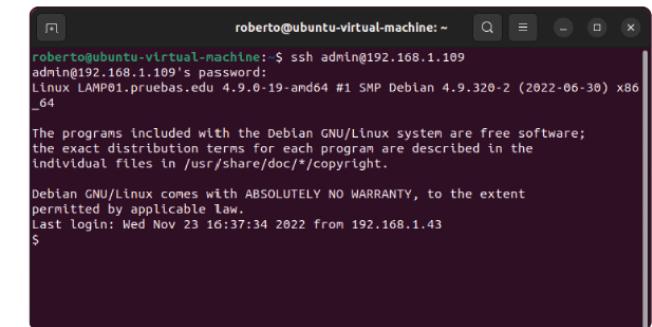
Para establecer la conexión se necesita el identificador del usuario con el que se va a conectar. Si la máquina **192.168.1.109** tuviera un usuario **admin** y un servidor SSH instalado, se podría iniciar la conexión de la siguiente forma:



```
roberto@ubuntu-virtual-machine:~$ ssh admin@192.168.1.109  
admin@192.168.1.109's password:
```

Ssh

El resultado permitirá obtener el control de la máquina 192.168.1.109:



```
roberto@ubuntu-virtual-machine:~$ ssh admin@192.168.1.109  
admin@192.168.1.109's password:  
Linux LAMP01.pruebas.edu 4.9.0-19-  
Debian GNU/Linux comes with ABSOLUTELY NO WARRANTY, to the extent  
Last login: Wed Nov 23 16:37:34 2022 from 192.168.1.43  
$
```

Obtener el control de la máquina



SSH es un protocolo que surge para proporcionar una forma de comunicación remota y segura entre equipos informáticos.



Aplicación práctica

Tiene acceso a una máquina con un sistema *Linux* y quiere administrar una jerarquía de directorios como la que se muestra más abajo en un sistema remoto. Sin embargo, tal jerarquía no existe en el disco duro de la máquina remota (192.168.1.109), pero existe una cuenta de usuario con las credenciales “*miusuario*, *mclave*”. Realice las llamadas al sistema adecuadas para conseguir crear la estructura de directorios en la máquina remota desde el sistema *Linux* que controla.

Jerarquía de directorios a crear:

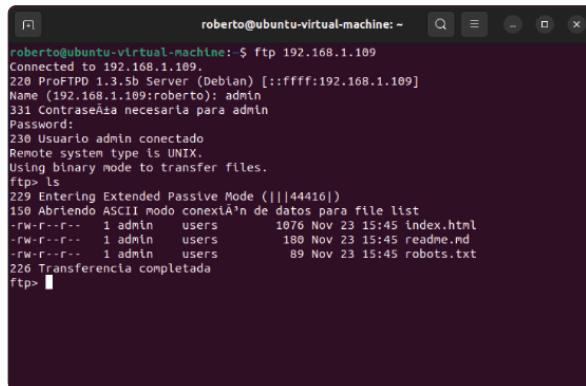
/prueba/segunda
/prueba/tercera
/prueba/cuarta

SOLUCIÓN

```
# ssh miusuario@192.168.1.109 mkdir /prueba  
mkdir /prueba/segunda  
mkdir /prueba/tercera  
mkdir /prueba/cuarta  
exit
```

ftp

ftp es el comando que se utiliza para establecer una comunicación con un servidor FTP. Este protocolo está especializado en la transferencia e intercambio de ficheros.



```
roberto@ubuntu-virtual-machine: ~ $ ftp 192.168.1.109  
Connected to 192.168.1.109.  
220 ProFTPD 1.3.5b Server (Debian) [::ffff:192.168.1.109]  
Name (192.168.1.109:roberto): admin  
331 Contraseña necesaria para admin  
Password:  
230 Usuario admin conectado  
Remote system type is UNIX.  
Using binary mode to transfer files.  
ftp> ls  
229 Entering Extended Passive Mode (|||44416|)  
150 Abriendo ASCII modo conexiÃ³n de datos para file list  
-r--r--r-- 1 admin users 1076 Nov 23 15:45 index.html  
-r--r--r-- 1 admin users 180 Nov 23 15:45 readme.md  
-r--r--r-- 1 admin users 89 Nov 23 15:45 robots.txt  
226 Transferencia completada  
ftp>
```

Ftp

En Windows el comando es el mismo:



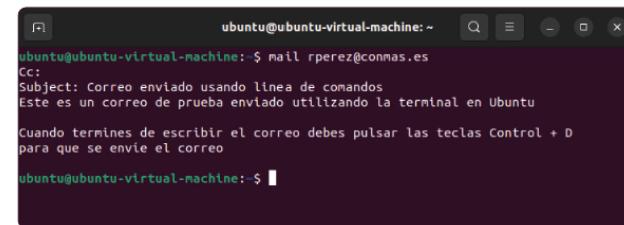
```
C:\Users\Taskkill /F /U :109  
Corriente: se envió la señal de término al proceso con PID 7196.  
C:\Users>ftp 192.168.1.109  
Conectado a 192.168.1.109.  
220 ProFTPD 1.3.5b Server (Debian) [::ffff:192.168.1.109]  
200 UTRX establecido en on  
Usuario (192.168.1.109:(none)): admin  
331 Contraseña necesaria para admin  
Contraseña:  
230 Usuario admin conectado  
Ftp> ls  
200 Comando PORT exitoso  
150 Abriendo ASCII modo conexiÃ³n de datos para file list  

```

Ftp en Windows

mail

Linux también permite enviar correos electrónicos a través de un comando del sistema. Se trata del comando **mail**. La forma es muy simple:



```
ubuntu@ubuntu-virtual-machine: ~ $ mail rerez@conmas.es  
Cc:  
Subject: Correo enviado usando linea de comandos  
Este es un correo de prueba enviado utilizando la terminal en Ubuntu  
Cuando termines de escribir el correo debes pulsar las teclas Control + D  
para que se envíe el correo  
ubuntu@ubuntu-machine: ~
```

Mail

En Windows no existe un comando similar que sea nativo. Si se quiere enviar un correo electrónico a través de la línea de comandos, se deben usar aplicaciones de terceros.

3.2. Uso de utilidades y comandos mediante lenguajes de script de uso común

Todo administrador de sistemas necesita herramientas para la configuración y automatización de las tareas repetitivas que requieren cierto tiempo de procesado. Los sistemas operativos modernos proporcionan estas herramientas en forma de lenguajes de *scripting*.

Los principales usos de este tipo de mecanismos son: automatización de copias de seguridad, posibilidad de apagar y reiniciar el sistema operativo cuando ocurre algún evento programable, automatización de operaciones de procesado, control y automatización de tareas administrativas, generación de conexiones de red de forma automática, etc.

Estos lenguajes no necesitan ser compilados y en la mayoría de los casos se apoyan en la línea de comandos para su despliegue.

Windows scripting

Windows mantiene desde sus orígenes el lenguaje batch como mecanismo de *scripting*. Se basa principalmente en ficheros de extensión ".bat" que contienen comandos internos mezclados con mecanismos de control de flujo de ejecución. El siguiente código es un ejemplo de fichero **batch** que realiza una cuenta atrás hasta que llega a 0.

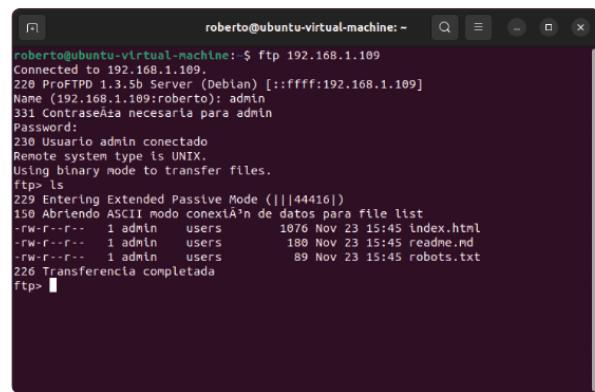
```
@echo off  
set a=11  
:loop  
set /A a-=1  
if %a% GTR 0 (  
echo quedan %a% segundos...  
timeout /t 1 /nobreak >nul  
goto loop  
)
```

SOLUCIÓN

```
# ssh miusuario@192.168.1.109 mkdir /prueba  
mkdir /prueba/segunda  
mkdir /prueba/tercera  
mkdir /prueba/cuarta  
exit
```

ftp

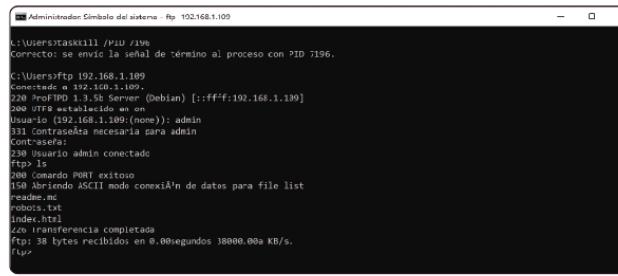
ftp es el comando que se utiliza para establecer una comunicación con un servidor FTP. Este protocolo está especializado en la transferencia e intercambio de ficheros.



```
roberto@ubuntu-virtual-machine:~$ ftp 192.168.1.109  
Connected to 192.168.1.109.  
220 ProFTPD 1.3.5b Server (Debian) [::ffff:192.168.1.109]  
Name (192.168.1.109:roberto): admin  
331 Contraseña necesaria para admin  
Password:  
230 Usuario admin conectado  
Remote system type is UNIX.  
Using binary mode to transfer files.  
ftp> ls  
229 Entering Extended Passive Mode (|||44416)|  
150 Abriendo ASCII modo conexiÃ³n de datos para file list  
-rw-r--r-- 1 admin users 1076 Nov 23 15:45 index.html  
-rw-r--r-- 1 admin users 180 Nov 23 15:45 readme.md  
-rw-r--r-- 1 admin users 89 Nov 23 15:45 robots.txt  
226 Transferencia completada  
ftp> 
```

Ftp

En Windows el comando es el mismo:



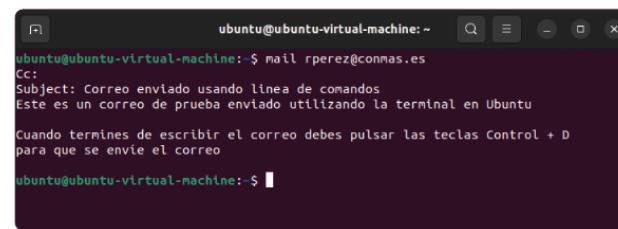
```
C:\Users\roberto>taskkill /PID 7196  
Correcto: se envío la señal de término al proceso con ?ID 7196.  
C:\Users\roberto>ftp 192.168.1.109  
Conectando al host 192.168.1.109...  
220 ProFTPD 1.3.5b Server (Debian) [:ffff:192.168.1.109]  
200 UTRF establecido en on  
User:admin (192.168.1.109:(none)): admin  
331 Contraseña necesaria para admin  
Password:  
230 Usuario admin conectado  
ftp> ls  
200 Comando PORT exitoso  
150 Abriendo ASCII modo conexiÃ³n de datos para file list  

```

Ftp en Windows

mail

Linux también permite enviar correos electrónicos a través de un comando del sistema. Se trata del comando **mail**. La forma es muy simple:



```
ubuntu@ubuntu-virtual-machine:~$ mail rperez@conmas.es  
Cc:  
Subject: Correo enviado usando lÃnea de comandos  
Este es un correo de prueba enviado utilizando la terminal en Ubuntu  
Cuando termines de escribir el correo debes pulsar las teclas Control + D  

```

Mail

En Windows no existe un comando similar que sea nativo. Si se quiere enviar un correo electrónico a través de la línea de comandos, se deben usar aplicaciones de terceros.

3.2. Uso de utilidades y comandos mediante lenguajes de script de uso común

Todo administrador de sistemas necesita herramientas para la configuración y automatización de las tareas repetitivas que requieren cierto tiempo de procesado. Los sistemas operativos modernos proporcionan estas herramientas en forma de lenguajes de *scripting*.

Los principales usos de este tipo de mecanismos son: automatización de copias de seguridad, posibilidad de apagar y reiniciar el sistema operativo cuando ocurre algún evento programable, automatización de operaciones de procesado, control y automatización de tareas administrativas, generación de conexiones de red de forma automática, etc.

Estos lenguajes no necesitan ser compilados y en la mayoría de los casos se apoyan en la línea de comandos para su despliegue.

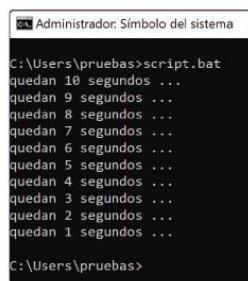
Windows scripting

Windows mantiene desde sus orígenes el lenguaje batch como mecanismo de *scripting*. Se basa principalmente en ficheros de extensión “.bat” que contienen comandos internos mezclados con mecanismos de control de flujo de ejecución. El siguiente código es un ejemplo de fichero **batch** que realiza una cuenta atrás hasta que llega a 0.

```
@echo off  
set a=11  
:loop  
set /A a=a-1  
if %a% GTR 0 (  
echo quedan %a% segundos...  
timeout /t 1 /nobreak >nul  
goto loop  
)
```

Los ficheros **batch** se denominan archivos de procesamiento por lotes. Esto es debido a que la ejecución siempre es secuencial.

Si se guarda este código en un fichero "script.bat", se podrá ejecutar desde la línea de comandos de la siguiente manera:



```
C:\Users\pruebas>script.bat
quedan 10 segundos ...
quedan 9 segundos ...
quedan 8 segundos ...
quedan 7 segundos ...
quedan 6 segundos ...
quedan 5 segundos ...
quedan 4 segundos ...
quedan 3 segundos ...
quedan 2 segundos ...
quedan 1 segundos ...

C:\Users\pruebas>
```

Ejecutar código



Actividades

11. ¿Cuál podría ser el motivo de que Microsoft mantenga el lenguaje de script basado en ficheros por lotes en los nuevos sistemas operativos? Justifique su respuesta.

Linux/Unix scripting

Linux/Unix utilizan el *scripting* con muchísima frecuencia y es la clave de muchos servidores que hoy en día existen en la web. Linux es un sistema operativo orientado a línea de comandos, como ya se sabe, y el mecanismo de *scripting* es algo que está implícito en la naturaleza de este tipo de sistemas operativos.

En Linux, existen varios tipos de intérpretes de comandos y, en función del que se utilice, el lenguaje de *scripting* cambia ligeramente. Algunos de estos intérpretes son: **bash**, **sh** y **csh**.



Definición

Scripting

Capacidad para poder ejecutar código interpretado que permita la administración del sistema operativo de forma automática.

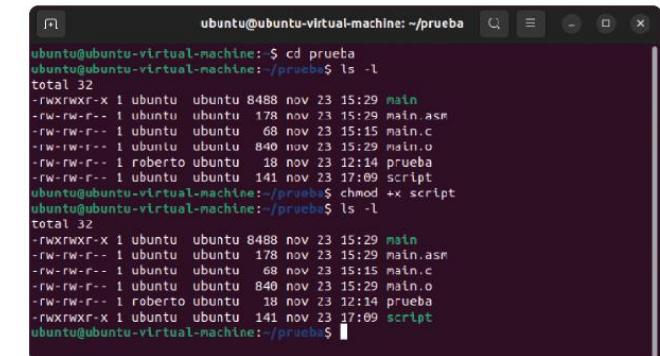
Un ejemplo de programa de *scripting* usando el intérprete **bash** sería:

```
#!/bin/bash
contador=10
while [ $contador -gt 0 ]
do
echo queda $contador segundos...
sleep 1
let contador=contador-1
done
```

Este código hace lo mismo que el que se ha visto en el apartado de *Windows scripting*.

Si se guarda el código en un fichero llamado **script**, se deben dar los siguientes pasos para ejecutarlo:

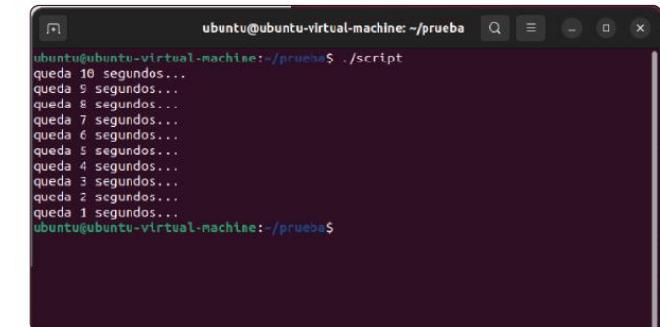
1. Asignar permisos de ejecución al *script*, usando el comando **chmod**:



```
ubuntu@ubuntu-virtual-machine: ~/prueba$ cd prueba
ubuntu@ubuntu-virtual-machine:~/prueba$ ls -l
total 32
-rwxrwxr-x 1 ubuntu  ubuntu 8488 nov 23 15:29 main
-rw-rw-r-- 1 ubuntu  ubuntu 178 nov 23 15:29 main.asm
-rw-rw-r-- 1 ubuntu  ubuntu 68 nov 23 15:15 main.c
-rw-rw-r-- 1 ubuntu  ubuntu 840 nov 23 15:29 main.o
-rw-rw-r-- 1 roberto  ubuntu 18 nov 23 12:14 prueba
-rw-rw-r-- 1 ubuntu  ubuntu 141 nov 23 17:09 script
ubuntu@ubuntu-virtual-machine:~/prueba$ chmod +x script
ubuntu@ubuntu-virtual-machine:~/prueba$ ls -l
total 32
-rwxrwxr-x 1 ubuntu  ubuntu 8488 nov 23 15:29 main
-rw-rw-r-- 1 ubuntu  ubuntu 178 nov 23 15:29 main.asm
-rw-rw-r-- 1 ubuntu  ubuntu 68 nov 23 15:15 main.c
-rw-rw-r-- 1 ubuntu  ubuntu 840 nov 23 15:29 main.o
-rw-rw-r-- 1 roberto  ubuntu 18 nov 23 12:14 prueba
-rwxrwxr-x 1 ubuntu  ubuntu 141 nov 23 17:09 script
ubuntu@ubuntu-virtual-machine:~/prueba$
```

Asignar permisos de ejecución al *script*

2. A continuación, se puede ejecutar el *script*, realizando la llamada al fichero que contiene el código:



```
ubuntu@ubuntu-virtual-machine:~/prueba$ ./script
queda 10 segundos...
queda 9 segundos...
queda 8 segundos...
queda 7 segundos...
queda 6 segundos...
queda 5 segundos...
queda 4 segundos...
queda 3 segundos...
queda 2 segundos...
queda 1 segundos...

ubuntu@ubuntu-virtual-machine:~/prueba$
```

Llamada al fichero que contiene el código

Los ficheros **batch** se denominan archivos de procesamiento por lotes. Esto es debido a que la ejecución siempre es secuencial.

Si se guarda este código en un fichero "script.bat", se podrá ejecutar desde la línea de comandos de la siguiente manera:

```
C:\Users\pruebas>script.bat
quedan 10 segundos ...
quedan 9 segundos ...
quedan 8 segundos ...
quedan 7 segundos ...
quedan 6 segundos ...
quedan 5 segundos ...
quedan 4 segundos ...
quedan 3 segundos ...
quedan 2 segundos ...
quedan 1 segundos ...

C:\Users\pruebas>
```

Ejecutar código



Actividades

11. ¿Cuál podría ser el motivo de que Microsoft mantenga el lenguaje de script basado en ficheros por lotes en los nuevos sistemas operativos? Justifique su respuesta.

Linux/Unix scripting

Linux/Unix utilizan el *scripting* con muchísima frecuencia y es la clave de muchos servidores que hoy en día existen en la web. Linux es un sistema operativo orientado a línea de comandos, como ya se sabe, y el mecanismo de *scripting* es algo que está implícito en la naturaleza de este tipo de sistemas operativos.

En Linux, existen varios tipos de intérpretes de comandos y, en función del que se utilice, el lenguaje de *scripting* cambia ligeramente. Algunos de estos intérpretes son: **bash**, **sh** y **csh**.



Definición

Scripting

Capacidad para poder ejecutar código interpretado que permita la administración del sistema operativo de forma automática.

Un ejemplo de programa de *scripting* usando el intérprete **bash** sería:

```
#!/bin/bash
contador=10
while [ $contador -gt 0 ]
do
echo queda $contador segundos...
sleep 1
let contador=contador-1
done
```

Este código hace lo mismo que el que se ha visto en el apartado de *Windows scripting*.

Si se guarda el código en un fichero llamado **script**, se deben dar los siguientes pasos para ejecutarlo:

1. Asignar permisos de ejecución al *script*, usando el comando **chmod**:

```
ubuntu@ubuntu-virtual-machine:~/prueba$ cd prueba
ubuntu@ubuntu-virtual-machine:~/prueba$ ls -l
total 32
-rwxrwxr-x 1 ubuntu  ubuntu 8488 nov 23 15:29 main
-rw-rw-r-- 1 ubuntu  ubuntu  178 nov 23 15:29 main.asm
-rw-rw-r-- 1 ubuntu  ubuntu   68 nov 23 15:15 main.c
-rw-rw-r-- 1 ubuntu  ubuntu  840 nov 23 15:29 main.o
-rw-rw-r-- 1 roberto  ubuntu  18 nov 23 12:14 prueba
-rw-rw-r-- 1 ubuntu  ubuntu 141 nov 23 17:09 script
ubuntu@ubuntu-virtual-machine:~/prueba$ chmod +x script
ubuntu@ubuntu-virtual-machine:~/prueba$ ls -l
total 32
-rwxrwxr-x 1 ubuntu  ubuntu 8488 nov 23 15:29 main
-rw-rw-r-- 1 ubuntu  ubuntu  178 nov 23 15:29 main.asm
-rw-rw-r-- 1 ubuntu  ubuntu   68 nov 23 15:15 main.c
-rw-rw-r-- 1 ubuntu  ubuntu  840 nov 23 15:29 main.o
-rw-rw-r-- 1 roberto  ubuntu  18 nov 23 12:14 prueba
-rwxrwxr-x 1 ubuntu  ubuntu 141 nov 23 17:09 script
ubuntu@ubuntu-virtual-machine:~/prueba$
```

Asignar permisos de ejecución al *script*

2. A continuación, se puede ejecutar el *script*, realizando la llamada al fichero que contiene el código:

```
ubuntu@ubuntu-virtual-machine:~/prueba$ ./script
queda 10 segundos...
queda 9 segundos...
queda 8 segundos...
queda 7 segundos...
queda 6 segundos...
queda 5 segundos...
queda 4 segundos...
queda 3 segundos...
queda 2 segundos...
queda 1 segundos...

ubuntu@ubuntu-virtual-machine:~/prueba$
```

Llamada al fichero que contiene el código



Aplicación práctica

En un sistema *Linux* es necesario realizar una tarea de administración que consiste en ejecutar un *script* que realiza ciertas operaciones. Sin embargo, el técnico responsable ha olvidado establecer los permisos adecuados al programa. El código se encuentra en el fichero “principal.sh” y está basado en bash. No tiene permisos de ejecución de root y pertenece al anterior administrador. Elabore la secuencia de comandos que hay que escribir en el terminal de *Linux* para que se pueda ejecutar con su cuenta, sabiendo que se ha accedido como root al sistema.

SOLUCIÓN

```
# chown root principal.sh  
# chmod +x principal.sh  
# ./principal.sh
```

4. Resumen

El administrador de un sistema operativo tiene a su disposición una gran cantidad de herramientas que puede utilizar para automatizar los distintos procesos. Estas herramientas forman la capa de interacción más cercana al usuario y permiten que este sea capaz de configurarlo para sus propósitos de procesamiento. Se han visto diferentes ejemplos de comandos que permiten un alto grado de configuración de un sistema operativo sin que para ello se tenga que ser tener conocimientos avanzados en programación o administración de sistemas.

Entre todas estas herramientas, destacan los lenguajes de *scripting*. La idea fundamental de estos lenguajes se ha conocido en apartados anteriores. Hoy en día, se han convertido en la pieza clave para proporcionar flexibilidad a la administración de todo tipo de servidores y son pocos los administradores que no hacen uso de ellos en algún momento.

Por último, destacar que todas estas herramientas se apoyan en una estructura más sólida y compacta que permite crear una consistencia en la ejecución de todo tipo de aplicaciones en un sistema operativo. Esta estructura está formada por las llamadas al sistema. Sin ellas, la ejecución de una aplicación en un sistema operativo sería inconsistente y muy poco robusta.