

**INSTITUTO POLITÉCNICO NACIONAL**

**ESCUELA SUPERIOR DE CÓMPUTO**

**Sistemas Operativos**

Práctica 2

**Entorno de Linux**

Integrantes:

|  |  |
| --- | --- |
| Juarez Leonel Reina Beatriz | 2CM14 |
| Martínez Ruiz Alfredo | 2CM14 |
| Méndez Castañeda Aurora | 2CM14 |
| Sandoval Acevedo Isaac | 2CV15 |

**Índice de Contenido**

[**Glosario de términos** 6](#_Toc66301531)

[**Investigación Previa** 7](#_Toc66301532)

[**Escritorios GNOME y KDE** 7](#_Toc66301533)

[**Entornos Command Line Interface y Graphical User Interface** 9](#_Toc66301534)

[**Terminal de Linux** 10](#_Toc66301535)

[**Tipos de usuario en Linux** 11](#_Toc66301536)

[**Direccionamiento relativo y absoluto** 11](#_Toc66301537)

[**Redireccionamiento** 13](#_Toc66301538)

[**Clasificación de los comandos en Linux** 15](#_Toc66301539)

[**Variables de entorno** 18](#_Toc66301540)

[**Desarrollo** 20](#_Toc66301541)

[**Códigos y ventanas de ejecución** 61](#_Toc66301542)

[1. Punto 11 del Desarrollo 61](#_Toc66301543)

[2. Punto 12 del Desarrollo 63](#_Toc66301544)

[**Conclusiones** 65](#_Toc66301545)

[**Bibliografía** 67](#_Toc66301546)

**Índice de Figuras**

[Imagen 1. Entorno de escritorio Gnome. 7](#_Toc66301416)

[Imagen 2. Entorno de escritorio KDE. 8](#_Toc66301417)

[Imagen 3. Entorno CLI y GUI. 10](#_Toc66301418)

[Imagen 4. Ejemplo de un directorio para ruta absoluta. 12](#_Toc66301419)

[Imagen 5. Ejemplo de un directorio para ruta relativa. 13](#_Toc66301420)

[Imagen 6. Comando ‘cal’ sin argumentos 20](#_Toc66301421)

[Imagen 7. Comando ‘cal’ con el argumento ‘y’ 20](#_Toc66301422)

[Imagen 8. Comando ‘date’ sin argumentos 21](#_Toc66301423)

[Imagen 9. Comando ‘date’ con argumentos para cambiar formato de fecha y hora 21](#_Toc66301424)

[Imagen 10. Argumentos que pueden acompañar al comando ‘date’ 21](#_Toc66301425)

[Imagen 11. Secuencias de ‘date’ para mostrar la salida en otro formato 22](#_Toc66301426)

[Imagen 12. Comando ‘echo’ con una cadena de caracteres 22](#_Toc66301427)

[Imagen 13. Comando ‘echo’ con variable creada y guardado de la variable en un archivo 23](#_Toc66301428)

[Imagen 14. Archivo guardado con ‘echo’ 23](#_Toc66301429)

[Imagen 15. Comando ‘time’ 23](#_Toc66301430)

[Imagen 16. Comando ‘uname’ sin argumentos 23](#_Toc66301431)

[Imagen 17. Comando ‘uname’ con el argumento ‘a’ 23](#_Toc66301432)

[Imagen 18. Argumentos que pueden acompañar al comando ‘uname’ 24](#_Toc66301433)

[Imagen 19. Comando ‘w’ sin argumentos 24](#_Toc66301434)

[Imagen 20. Argumentos que pueden acompañar al comando ‘w’ 24](#_Toc66301435)

[Imagen 21. Comando ‘who’ sin argumentos 24](#_Toc66301436)

[Imagen 22. Argumentos que pueden acompañar al comando ‘who’ 25](#_Toc66301437)

[Imagen 23. Comando ‘bash’ sin argumentos 25](#_Toc66301438)

[Imagen 24. Versión de bash 25](#_Toc66301439)

[Imagen 25. Argumentos que pueden acompañar al comando ‘bash’ 26](#_Toc66301440)

[Imagen 26. Comando ‘wc’ 26](#_Toc66301441)

[Imagen 27. Argumentos que pueden acompañar al comando ‘wc’ 26](#_Toc66301442)

[Imagen 28. Comando ‘apt-get’ 27](#_Toc66301443)

[Imagen 29. Antes de usar el comando clear. 27](#_Toc66301444)

[Imagen 30. Después de usar el comando clear. 28](#_Toc66301445)

[Imagen 31. Comando ‘ifconfig’ 28](#_Toc66301446)

[Imagen 32. Comando ‘df’ 28](#_Toc66301447)

[Imagen 33. Comando ‘du’ 29](#_Toc66301448)

[Imagen 34. Uso del comando ‘du -bsh’ 29](#_Toc66301449)

[Imagen 35. Comando ‘du -sm \*’ 29](#_Toc66301450)

[Imagen 36. Comando du con X directorios más pesados. 30](#_Toc66301451)

[Imagen 37. Comando du con X directorios más pesados representados en MB. 30](#_Toc66301452)

[Imagen 38. Comando ‘ps’ sin argumentos. 30](#_Toc66301453)

[Imagen 39. Comando ‘ps’ con argumento –a. 31](#_Toc66301454)

[Imagen 40. Comando ‘ps’ con argumento –g. 31](#_Toc66301455)

[Imagen 41. Conociendo el PID a aniquilar. 32](#_Toc66301456)

[Imagen 42. Comando ‘kill -9’ 32](#_Toc66301457)

[Imagen 43. Comando ‘trap -l’ 33](#_Toc66301458)

[Imagen 44. Comando ‘pwd’. 33](#_Toc66301459)

[Imagen 47. Uso de apt en la terminal de Ubuntu. 36](#_Toc66301460)

[Imagen 48. Uso de ps con el parámetro “u” en la terminal de Ubuntu. 36](#_Toc66301461)

[Imagen 49. Uso de ps con el parámetro “-fea” en la terminal de Ubuntu. 37](#_Toc66301462)

[Imagen 50. Uso de man con el parámetro “clear” en la terminal de Ubuntu. 37](#_Toc66301463)

[Imagen 51. Uso de cat para crear un archive .txt en la terminal de Ubuntu. 38](#_Toc66301464)

[Imagen 52. Uso de fg en la terminal de Ubuntu. 38](#_Toc66301465)

[Imagen 53. Uso de cd en la terminal de Ubuntu. 38](#_Toc66301466)

[Imagen 54. Uso de ls con el parámetro -l en la terminal de Ubuntu. 39](#_Toc66301467)

[Imagen 55. Ejemplo del uso del comando ls -la. 39](#_Toc66301468)

[Imagen 56. Ejemplo usando rm –interactive. 40](#_Toc66301469)

[Imagen 57. Ejemplo del uso de rm -r. 40](#_Toc66301470)

[Imagen 58. Directorios antes de mover. 41](#_Toc66301471)

[Imagen 59. Directorios después de moverlos. 41](#_Toc66301472)

[Imagen 60. Ejemplo de mkdir -p. 42](#_Toc66301473)

[Imagen 61. Visualización del editor de nano(pico). 43](#_Toc66301474)

[Imagen 62. Visualización de Vi. 43](#_Toc66301475)

[Imagen 63. Uso del comando pwd 44](#_Toc66301476)

[Imagen 64. Ejemplo 1 de direccionamiento absoluto. 44](#_Toc66301477)

[Imagen 65. Ejemplo 2 de direccionamiento absoluto. 44](#_Toc66301478)

[Imagen 66. Ejemplo 3 de direccionamiento absoluto. 44](#_Toc66301479)

[Imagen 67. Ejemplo 1 de direccionamiento relativo. 44](#_Toc66301480)

[Imagen 68. Ejemplo 2 de direccionamiento relativo. 45](#_Toc66301481)

[Imagen 69. Ejemplo 3 de direccionamiento relativo. 45](#_Toc66301482)

[Imagen 70. Ejemplo de comando rm usando \* para eliminar todos los archivos. 45](#_Toc66301483)

[Imagen 71. Ejemplo de comando rm usando \* para eliminar todas las carpetas. 45](#_Toc66301484)

[Imagen 72. Ejemplo de comando rm usando \* para eliminar archivos de una carpeta hija 46](#_Toc66301485)

[Imagen 73. Ejemplo de comando rm usando “?” al final. 46](#_Toc66301486)

[Imagen 74. Ejemplo de comando rm usando “?” para eliminar todas las carpetas. 46](#_Toc66301487)

[Imagen 75. Ejemplo de comando rm usando “?” para eliminar carpetas hijas con el mismo formato que las carpetas madre. 46](#_Toc66301488)

[Imagen 76. Uso de ls con el parámetro -l y con un redireccionamiento a un archivo. 47](#_Toc66301489)

[Imagen 77. Captura del archivo lista.txt 47](#_Toc66301490)

[Imagen 78. Captura del archivo lista.txt actualizado. 47](#_Toc66301491)

[Imagen 81. Uso del comando mv para mover un archivo a una carpeta. 48](#_Toc66301492)

[Imagen 82. Directorio actualizado. 49](#_Toc66301493)

[Imagen 83. Captura final del directorio después del comando mv para mover un directorio. 49](#_Toc66301494)

[Imagen 84. Captura final del directorio después del comando mv para mover varios archivos. 49](#_Toc66301495)

[Imagen 85. Uso de cp para copiar un archivo a otro directorio. 50](#_Toc66301496)

[Imagen 86. Directorio de documentos. 50](#_Toc66301497)

[Imagen 87. Captura final del directorio. 50](#_Toc66301498)

[Imagen 88. Comando para crear usuario. 51](#_Toc66301499)

[Imagen 89. Comando para verificar un usuario. 51](#_Toc66301500)

[Imagen 90. Inicio de sesión del nuevo usuario. 51](#_Toc66301501)

[Imagen 91. Eliminación del nuevo usuario. 51](#_Toc66301502)

[Imagen 92. Creación de archivo con nano. 52](#_Toc66301503)

[Imagen 93. Código de ‘Hola mundo’ en nano. 52](#_Toc66301504)

[Imagen 94. Compilación de ‘HolaMundo.c’. 52](#_Toc66301505)

[Imagen 95. Ejecución de ‘HolaMundo’. 52](#_Toc66301506)

[Imagen 96. Creación de archivo con vi. 52](#_Toc66301507)

[Imagen 97. Código de ‘Hola mundo’ en vi. 53](#_Toc66301508)

[Imagen 98. Visualización del archivo ‘HolaMundo.c’ en vi. 53](#_Toc66301509)

[Imagen 99. Compilación de ‘HolaMundo.c’. 53](#_Toc66301510)

[Imagen 100. Ejecución de ‘HolaMundo’. 53](#_Toc66301511)

[Imagen 101. Intentó de instalación de pico 53](#_Toc66301512)

[Imagen 102. Código de uso de la función getenv(). 54](#_Toc66301513)

[Imagen 103. Compilación y ejecución del archivo ‘Programa1a.c’. 54](#_Toc66301514)

[Imagen 104. Código donde se piden al usuario los 10 números. 54](#_Toc66301515)

[Imagen 105. Código donde se llama a la función archivo. 55](#_Toc66301516)

[Imagen 106. Código donde abrimos y escribimos los 10 números. 55](#_Toc66301517)

[Imagen 107. Código donde cerramos el archivo. 55](#_Toc66301518)

[Imagen 108. Compilación, ejecución, inserción de números y visualización del archivo. 56](#_Toc66301519)

[Imagen 109. Ejecución del Programa2a.c 57](#_Toc66301520)

[Imagen 110.El archivo “numeros.txt. 57](#_Toc66301521)

[Ahora si ejecutamos el Programa3a.c obtenemos lo siguiente: 57](#_Toc66301522)

[Imagen 111. Ejecución del Programa3a.c 58](#_Toc66301523)

[Imagen 112. Solver para desviación estandar. 59](#_Toc66301524)

[Ahora explicaré un poco más a detalle el código: 59](#_Toc66301525)

[Imagen 113. Programa3a.c primera parte. 59](#_Toc66301526)

[Imagen 114. Programa3a.c segunda parte. 60](#_Toc66301527)

[Algo interesante que quiero comentar es que al utilizar funciones de la librería math.h me salía el siguiente error: 60](#_Toc66301528)

[Imagen 115. Error en Linux con la librería math.h 60](#_Toc66301529)

[Imagen 116. Arreglo del problema con math.h 61](#_Toc66301530)

# **Glosario de términos**

FreeBDS: FreeBSD es un sistema operativo de código abierto para computadoras basado en las CPU de arquitectura x86, Intel 80386, Intel 80486, y Pentium.

KDE: Es una comunidad internacional que desarrolla software libre.

Framework: Un entorno de trabajo​, o marco de trabajo​ es un conjunto estandarizado de conceptos, prácticas y criterios para enfocar un tipo de problemática particular que sirve como referencia, para enfrentar y resolver nuevos problemas de índole similar.

GTK+: The GIMP Toolkit​​ es una biblioteca de componentes gráficos multiplataforma para desarrollar interfaces gráficas de usuario (GUI).

Kernel: El núcleo o kernel es la parte central de un sistema operativo y es el que se encarga de realizar toda la comunicación segura entre el software y el hardware del ordenador.

PATH: Variable de entorno de los sistemas operativos POSIX y los sistemas de Microsoft, en ella se especifican las rutas en las cuales el intérprete de comandos debe buscar los programas a ejecutar.

Proceso: Sucesión de instrucciones que pretenden llegar a un estado final o que persiguen realizar una tarea concreta.

Comando: Mensaje enviado al ordenador, por parte del usuario, y que va a provocar una respuesta en este.

Compilar: Traductor que transforma un programa entero de un lenguaje de programación a otro, usualmente el lenguaje objetivo es código máquina.

Ejecutar: Proceso mediante el cual una computadora lleva a cabo las instrucciones de un programa informático.

Shell: Programas que proveen una interfaz de usuario para acceder a los servicios del sistema operativo.

Directorio: Contenedor virtual en el que se almacenan una agrupación de archivos informáticos y otros subdirectorios, atendiendo a su contenido, a su propósito o a cualquier criterio que decida el usuario

Qt: Framework multiplataforma orientado a objetos ampliamente usado para desarrollar programas (software) que utilicen interfaz gráfica de usuario

Konsole: Emulador de terminal, desarrollado para el proyecto de escritorio libre KDE.

GCC: La sigla GCC significa "GNU Compiler Collection". Originalmente significaba "GNU C Compiler"; todavía se usa GCC para designar una compilación en C. G++ refiere a una compilación en C++.

Konsole: Es un emulador de terminal, desarrollado para el proyecto de escritorio libre KDE. Está altamente integrado con el entorno, y soporta sesiones múltiples en pestañas, operaciones de edición, favoritos, impresión, etc.

Comodines: Serie de elementos que podemos incorporar a la web y que usará el motor del buscador (o también conocido como araña) para encontrar una respuesta más certera a nuestra consulta.

# **Investigación Previa**

## **Escritorios GNOME y KDE**

Cuando se trabaja en el sistema operativo Linux, es importante escoger el entorno de escritorio que mejor se adapte al usuario final y que, sobre todo, ofrezca el mejor rendimiento del equipo en el que se va a instalar.

**Escritorio Gnome**

Gnome, es un tipo de entorno de escritorio, que además es un completo sistema de desarrollo para los sistemas operativos basados en Unix, por ejemplo, Linux, FreeBDS y Solaris.

El proyecto Gnome, **fue iniciado por los desarrolladores mexicanos Federico Mena y Miguel de Icaza** y actualmente se encuentra disponible en un poco más de 48 idiomas. Gnome, nace como una alternativa completamente libre al famoso KDE.

Características de Gnome:

* Entorno limpio y claro con diseño minimalista.
* Facilidad de uso con una muy pequeña curva de aprendizaje si es la primera vez que trabajas con él.
* Todavía en evolución. El concepto de Gnome se está readaptando por completo a la nueva generación de pantallas y dispositivos, por tanto, hubo, hay y habrá cambios sustanciales en el mismo.
* La mayor parte del código está escrito en C y para las aplicaciones se utilizan las herramientas GTK+, a pesar de que pueden usarse otros muchos lenguajes.



Imagen 1. Entorno de escritorio Gnome.

**Escritorio KDE**

KDE**,** es uno de los entornos de escritorio más populares para el sistema operativo Linux. Cuenta con una interesante interfaz, muy amigable y atractiva para usuarios que migran de otros sistemas operativos como Windows, siendo una de las características principales por las que es uno de los entornos de escritorio más populares en las distribuciones Linux.

KDE, es un proyecto internacional de software libre, orientado específicamente a desarrollar un entorno de escritorio e infraestructura de desarrollo para los sistemas operativos GNU/Linux, Mac OSX e incluso, Windows**.**

Una de las ventajas de KDE, es que está hecho para ser [**personalizado**](https://culturacion.com/como-personalizar-una-carpeta-en-windows/). Es por esta razón, que viene con una gran cantidad de funciones específicas que ofrecen la posibilidad de adaptar el aspecto visual y el desempeño del escritorio, a los requerimientos del sistema y del usuario.



Imagen 2. Entorno de escritorio KDE.

Características de KDE:

* Es un entorno muy sofisticado y actual en constante evolución.
* Facilidad de uso y personalización.
* Escrito en C++ y no demasiado exigente en recursos.
* Con arquitectura modular en la que se pueden distinguir distintos frameworks dedicados a tareas específicas y muy variadas.
* Dispone de cientos de aplicaciones exclusivas que están programadas usando las librerías propias de KDE y Qt.
* Es un entorno bastante estable y elegante.

## **Entornos Command Line Interface y Graphical User Interface**

**La Interfaz de Línea de Comandos o CLI**

La interfaz de línea de comandos, o CLI, por sus siglas en inglés (*command-line interface*) es un método de comunicación entre usuario y máquina que acepta instrucciones del usuario a través de líneas de texto (siguiendo unas determinadas reglas de sintaxis que puedan ser interpretadas por el sistema operativo).

Es el método que se utilizaba antes de la aparición de las interfaces gráficas o *GUI*, que sirvió para acercar la informática al gran público, al permitir una interacción del usuario con la maquina mucho más amigable o intuitiva (a costa, obviamente, de una mayor complejidad y consumo de recursos).

La línea de comandos es algo que está muy impregnado en la cultura “linuxera”. Y es que, en Linux, al igual que puedes optar por utilizar diferentes entornos de escritorio, también tienes la posibilidad de no disponer de ninguno, y **operar directamente a través de la interfaz de línea de comandos** (de hecho, algunas distribuciones, como la versión de Ubuntu para servidores, vienen de serie así).

**Graphical User Interface o GUI**

Una Graphical User Interface o GUI es una interfaz que se puede utilizar para controlar PC, tabletas y otros dispositivos. Las GUI utilizan elementos gráficos como iconos, menús e imágenes para facilitar el manejo del usuario humano. Tanto los sistemas operativos como las aplicaciones utilizan una interfaz gráfica de usuario. De hecho, casi todos los programas para usuarios finales de hoy en día vienen con esta interfaz.

Suele estar basada en la interacción a través del ratón y el teclado (aunque el control a través de gestos es cada vez más común). La señal del dispositivo se transmite al ordenador, que luego la traduce en un movimiento equivalente en la pantalla. Por ejemplo, si un usuario hace clic en un determinado icono de programa en el menú, se ejecuta la instrucción correspondiente y se abre el programa.

La GUI es, por lo tanto, una especie de **traductor** en la comunicación entre el humano y la máquina. Sin la GUI, tendrías que utilizar la línea de comandos para llamar a programas y aplicaciones.

Los componentes más comunes de una GUI son los siguientes:

* Campos de entrada.
* Ventana.
* Lienzo (*canvas*).
* Marcos.
* Botones.
* Bloques de encabezado.
* Campos de texto.

Es importante destacar que cuando hablamos de CLI o GUI nos estamos refiriendo a métodos de interacción, no una herramienta o programa informática. De hecho, a la herramienta que posibilita cualquiera de estas formas de interacción se la suele conocer como ***shell***.

Una shell es una herramienta que acepta ordenes o instrucciones por parte del usuario y ejecuta operaciones. En Linux podemos interactuar con **multitud de shells diferentes**, tanto a nivel de interfaz de línea de comandos, como a nivel de interfaz gráfica.

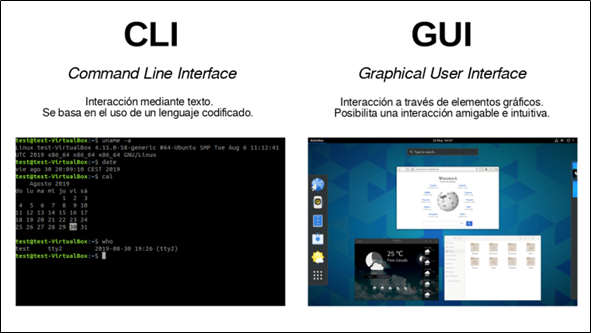


Imagen 3. Entorno CLI y GUI.

## **Terminal de Linux**

Linux utiliza una herramienta llamada consola de comandos o Terminal desde la cual se pueden ejecutar una serie de sentencias para acceder a ciertos lugares del sistema, realizar ciertas tareas o simplemente visualizar reportes importantes.

Un terminal es una forma de acceder al sistema sin utilizar la interfaz gráfica, es decir, realizar todo tipo de tareas en formato texto. La forma de utilizar el sistema de este modo es mediante órdenes. Es utilizada para permitir a los usuarios más avanzados y técnicos controlar hasta el más mínimo detalle del sistema operativo.

El terminal muestra en pantalla un indicador de línea de órdenes esperando que el usuario introduzca una orden. Este indicador finaliza generalmente por un carácter $, cuando eres un usuario normal, o # cuando eres un súper usuario (administrador).

Desde esta consola se puede ejecutar todo tipo de binarios, aunque se apropia en ejecutar aquellos que no tienen interfaz gráfica y que deben usarse mediante comandos. Igual que en otros sistemas operativos, se puede ejecutar cualquier binario o script directamente desde su directorio. Además, Linux tiene también un PATH donde se pueden guardar binarios y ejecutarlos sin tener que desplazarnos hasta su directorio.

La terminal de Linux se basa en un lenguaje de scripting conocido como Bash, heredada de sh, la consola de Unix. Podemos ejecutar scripts desde la consola, ejecutar binarios y realizar todo tipo de tareas. A diferencia de Windows, Linux cuenta en su terminal una gran cantidad de herramientas muy avanzadas para administrar y controlar el sistema operativo.

Todo lo que hacemos desde una interfaz, podríamos hacerlo perfectamente desde la terminal. Aunque de forma más complicada, larga y menos intuitiva.

## **Tipos de usuario en Linux**

Linux es un sistema multiusuario, por lo tanto, la tarea de añadir, modificar, eliminar y en general administrar usuarios se convierte en algo no solo rutinario, sino importante, además de ser un elemento de seguridad que mal administrado, podría convertirse en un enorme problema de seguridad.

Cada usuario posee un directorio personal o directorio HOME en el que se almacena la configuración de su entorno o perfil, así como los datos personales de dicho usuario. Por defecto, a ese directorio sólo tendrá acceso el mismo usuario, a no ser que él específicamente conceda permisos a otros usuarios para poder acceder.

Los usuarios y grupos creados se identifican en el sistema por un número, único para cada uno de ellos, y no por el nombre que se le ponga al usuario o al grupo. Dichos números los asigna el sistema automáticamente.

Se pueden identificar tres tipos de usuarios en Linux:

* *Usuario ROOT*

También llamado superusuario o administrador. Es el único usuario con privilegios sobre el control total del sistema, por tanto, es el único que tiene derecho a administrar las cuentas de usuario del sistema.

La ruta del directorio personal es */root,* además que su identificador de usuario (UID) es 0.

Cabe decir que este usuario es el único que puede establecer los identificadores de usuario manualmente.

* *Usuarios normales o finales*

Son los usuarios habituales del sistema, que utilizarán los recursos de este. Cada usuario sólo podrá personalizar su entorno de trabajo.

La ruta del directorio personal es */home/<usuario>,* por otro lado, su UID es superior a 1000.

* *Usuarios especiales o de sistema*

Son incorporados por el propio sistema, y se encargan de administrar los demonios (programa que se ejecuta en segundo plano) del mismo. Estos usuarios no pueden iniciar sesión en el sistema ni tener un shell donde trabajar, por tanto, no tienen contraseña asignada.

No poseen un directorio personal y su identificador de usuario se encuentra entre 1 y 100.

Linux permite la creación de grupos de usuarios, para poder establecer permisos y restricciones por grupos en lugar de por usuarios individuales. Por defecto, cada usuario pertenecerá a un grupo que se crea de forma automática con su mismo nombre de usuario. Será el usuario root el que pueda asignar los usuarios a los distintos grupos del sistema.

## **Direccionamiento relativo y absoluto**

Primero definamos a lo que nos referimos con las rutas, dado que es importante definirlo en este punto…

En definición, las rutas sirven para indicar la localización exacta de un archivo o carpeta, siguiendo una estructura de directorios o árbol de directorios. Todos los sistemas operativos disponen de un sistema de archivos en el que podemos emplear las rutas para localizar un archivo o carpeta. En muchas ocasiones se emplea como sinónimo de ruta la palabra «path» (ruta en inglés) y en algunas ocasiones se le llama simplemente directorio.

Existen dos tipos de rutas de las cuales veremos más detalles a continuación:

* **Rutas absolutas**

Una ruta absoluta se representa la ruta completa del recurso. Dicho de otra forma, se parte desde el directorio raíz hasta llegar al recurso.

El sistema de ficheros es una estructura jerárquica que, en el caso de Linux, tiene una raíz que se indica cuando se pone solamente el carácter barra “/”. Cuando yo quiero indicar dónde se encuentra un elemento usando una ruta absoluta, siempre tendré que indicarle todos los directorios por los que hay que pasar empezando desde la raíz del sistema.

A continuación, veremos una imagen sencilla donde queremos editar documento1.txt con ayuda de la ruta absoluta: nano /home/usuario/carpeta/carpeta3/documento1.txt, donde nano sirve para abrir un editor de texto que trae la terminal.

Imagen 4. Ejemplo de un directorio para ruta absoluta.

Como vemos, tenemos que escribir toda la ruta del archivo partiendo desde la raíz.

* **Rutas relativas**

En el caso de las rutas relativas se representa sólo una parte de la ruta, esto quiere decir que no es necesario tomar en cuenta desde la raíz. Esto es posible porque en las rutas relativas se tiene en cuenta el directorio actual de trabajo. Dicho de otro modo, las rutas relativas indican el camino para encontrar un elemento, desde el directorio en donde nos encontramos posicionados.

Para saber si son correctas o no tenemos que saber siempre desde dónde se han utilizado.

A continuación, veremos el mismo directorio que vimos en el punto de rutas absolutas y plantearemos unos ejemplos.

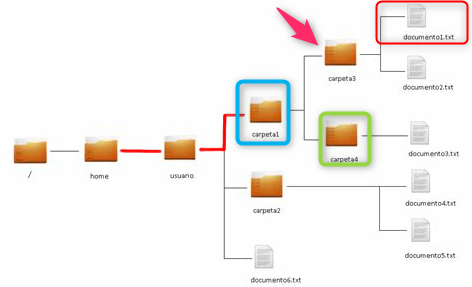


Imagen 5. Ejemplo de un directorio para ruta relativa.

Supongamos que nos encontramos en la carpeta 3, vamos a plantear los siguientes ejemplos:

* + Si queremos editar el documento1.txt es tan sencillo como escribir el siguiente comando: nano documento1.txt
  + Ahora, si queremos movernos hacia la carpeta 4: cd ../carpeta4

Esto lo hacemos porque cuando escribimos los dos puntos nos referimos a que vamos a la carpeta padre, en este caso la carpeta1 y después accedemos desde ahí a la carpeta4, podríamos ver de otra forma el comando, dividido en dos comandos: cd .. y cd carpeta4

* + Si quisiéramos tan sólo ir a usuario tendríamos que ejecutar el siguiente comando: cd ../..

Como vemos, podemos acceder a un archivo o directorio de una manera bastante sencilla, aunque claro cada una tiene sus ventajas y caso de uso, podríamos decir que es recomendable usar rutas absolutas cuando se trata de un directorio o archivo que siempre estará en el mismo lugar, mientras que podemos usar las rutas relativas en todos los casos donde sabemos en qué directorio nos encontramos y para movernos a través de los sistemas de archivos más fácilmente.

## **Redireccionamiento**

Para esta parte debemos comenzar por definir 3 conceptos básicos…

* **La entrada estándar**

Representa los datos que son ingresados para el correcto funcionamiento de una aplicación. Un ejemplo de ellos puede ser un archivo o información ingresada desde la terminal.

En la terminal se representa como el tipo 0.

* **La salida estándar**

Es el medio que utiliza una aplicación para mostrar información de sus procesos y/o resultados, estos pueden ser simples mensajes o archivos con datos.

En la terminal se representa como el tipo 1.

* **El error estándar**

Es la manera en que las aplicaciones nos informan sobre los problemas que pueden ocurrir al momento de su ejecución.

Se representa como el tipo 2 en la terminal.

Ahora que ya tenemos esos conceptos básicos podemos ser capaces de concebir la definición de redireccionamiento, y podemos decir que las redirecciones consisten en trasladar información de un tipo a otro (los tipos mencionados en los puntos superiores), por ejemplo, de error estándar a la salida estándar o de la salida estándar a la entrada estándar a través de la terminal, y lo hacemos usando el símbolo **>.**

La mejor forma de entender este concepto es a través de ejemplos, por ello veremos algunos a continuación:

1. Para hacer la redirección de la salida de un comando y enviarla a un archivo txt pero reemplazando toda la información que hay en el archivo: ls > archivo.txt
2. Ahora bien, para hacer lo mismo del punto anterior pero en vez de sobreescribir el archivo, queremos añadir la información al archivo: ls >> archivo.txt
3. Si quisiéramos redireccionar los errores que nos muestra una aplicación a una salida estándar: aplicacion1 2 >> &1, siendo 2 el error y &1 la salida estándar.
4. Ahora bien, podemos redireccionar las entradas también, para ello hacemos uso del operador **<** como el ejemplo que veremos a continuación: echo “hola” > saludo.txt y cat < saludo.txt

Ya que entendimos un poco mejor la definición de estos términos, vamos a ver ahora qué son las tuberías.

Las tuberías son un tipo especial de redirección que permiten enviar la salida estándar de un comando como entrada estándar de otro. La forma de representarlo es con el símbolo **| (pipe)**, o visto de otra forma: **<comando1> | <comando2>.**

Su principal utilidad es que nos ofrece la posibilidad de concatenar comandos, enriqueciendo la programación.

Vemos unos ejemplos a continuación:

1. Para ordenar las palabras de un archivo por orden alfabético: cat archivo.txt | sort
2. Para contar las líneas de un archivo por orden alfabético: cat archivo.txt | wc -l

Y así como los ejemplos anteriores podemos encontrar múltiples formas de combinar los comandos sin límites.

## **Clasificación de los comandos en Linux**

Como la mayoría de sistemas operativos modernos, Linux también le ofrece al usuario dos posibilidades de interacción: todos los ajustes que se realizan en la interfaz gráfica (GUI) se pueden llevar a cabo con **líneas de comandos** en el denominado **shell**.

* Comandos básicos de Linux

En esta categoría se incluyen todas las órdenes fundamentales que se utilizan para administrar el terminal, con las que se puede limpiar la ventana de la consola, recuperar comandos anteriores desde la historia o finalizar la sesión.

* + clear
  + exit
  + help
  + history
* Páginas de ayuda

Linux ofrece en el terminal páginas de ayuda y de documentación, las cuales contienen una exhaustiva descripción de todos los programas, comandos, archivos de configuración, formatos de archivo y funciones del kernel.

* + apropos
  + info
  + man
  + pinfo
* Operaciones en el directorio

Algunos comandos básicos de Linux permiten llevar a cabo operaciones en los directorios del sistema, como puede ser crear ficheros, borrarlos y gestionarlos, así como navegar por el árbol del directorio.

* + cd
  + chroot
  + ls
  + mkdir
  + pwd
  + rmdir.
* Operaciones en archivos

Los comandos Linux de esta categoría permiten llevar a cabo diversas operaciones desde el terminal que atañen a los archivos y se pueden copiar, desplazar, renombrar o borrar archivos del sistema.

* + basename
  + cat
  + cmp
  + cp
  + dirname
  + file
  + lsof
  + mv
  + shred
  + sort
  + split
  + stat
  + touch
  + uniq
* Gestión de los permisos

Con Linux se pueden definir los derechos de acceso y de posesión de archivos y ficheros por parte de los usuarios fácilmente.

* + chattr
  + chgrp
  + chmod
  + chown
  + lsattr
* Opciones de búsqueda

Linux también dispone de diversos comandos para explorar el sistema desde el terminal.

* + find
  + grep
  + locate
  + tre-agrep
  + updatedb
  + whereis
  + which
* Información sobre los usuarios

Utiliza los programas de esta categoría para solicitar información detallada sobre los usuarios registrados en el sistema, así como sobre sus grupos y procesos.

* + finger
  + groups
  + id
  + last
  + w
  + who
  + whoami.
* Gestión de cuentas de usuario

Linux ofrece toda una serie de programas con los cuales se pueden crear, eliminar y gestionar cuentas de usuario y grupos a través del terminal. Estos aparecen en el siguiente listado general de comandos de Linux para la administración de cuentas de usuario, categoría en la que también se incluyen las órdenes que permiten abrir programas con los derechos de otros usuarios, incluidos los de administrador.

* + adduser
  + chfn
  + chsh
  + deluser
  + groupadd
  + delgroup
  + groupmod
  + newgrp
  + passwd
  + sudo
  + su
  + usermod
* Gestión del sistema

En esta categoría desciframos los comandos básicos de Linux para la administración del sistema. Con ellos puedes, por ejemplo, reiniciar o apagar el sistema desde la consola e incluso programar estas acciones. La mayor parte de comandos de gestión del sistema requiere permisos de administrador.

* + logger
  + reboot
  + rtcwake
  + shutdown
* Información del sistema

Esta categoría recoge todos los programas de líneas de comandos con los cuales se solicitan información y mensajes de estado del sistema para hacerse una idea global de su estado general.

* date
* df
* dmesg
* du
* free
* hostname
* uname
* uptime
* Información acerca del hardware

Los comandos Linux englobados en esta categoría entregan datos detallados sobre los componentes de hardware sobre los que se ejecuta el sistema.

* lscpu
* lshw
* lspci
* lsusb
* Gestión de procesos

En Linux se denominan procesos a las instancias de un programa en ejecución. Los siguientes comandos pertenecen al repertorio estándar de la gestión de procesos y permiten supervisarlos cómodamente desde el terminal e intervenir si es necesario.

* chrt
* ionice
* kill
* killall
* nice
* nohup
* pgrep
* pidof
* pkill
* ps
* pstree
* renice
* sleep
* taskset
* top
* Editores

Con Linux no es necesario utilizar un programa de edición de textos para modificar archivos de configuración, editar fragmentos de código o escribir notas cortas, pues es posible abrir editores de texto simples en el terminal.

* emacs
* nano
* vim
* Gestión de redes

También es posible gestionar redes en el terminal utilizando Linux. Ya se trate de examinar una conexión, de solicitar información al DNS, de configurar interfaces o de enviar archivos a otro ordenador.

* + arp
  + dig
  + ftp
  + ip
  + iw
  + netstat
  + nslookup
  + ping
  + route
  + rsync
  + sftp
  + scp
  + traceroute
  + tty

## **Variables de entorno**

Las variables de entorno son valores dinámicos que afectan los programas o procesos que se ejecutan en un servidor. Existen en todos los sistemas operativos y su tipo puede variar. Las variables de entorno se pueden crear, editar, guardar y eliminar. En Linux, las variables de entorno son marcadores de posición para la información almacenada dentro del sistema que pasa datos a los programas iniciados en shells (intérpretes de comando) o sub-shells.

Para poder ver la lista completa de variables de entorno de tu versión de Linux utilizando el comando printenv. El uso simple de este en Ubuntu proporcionará un gran resultado que muestra todas las variables.

Se puede obtener una salida más manejable agregando detalles en la línea de comando, como, por ejemplo:

printenv | les

Las variables de entorno suelen estar en mayúsculas, aunque también puedes crear variables de entorno en minúsculas. La salida de printenv muestra todas las variables de entorno en mayúsculas. Una cosa importante a tener en cuenta es que las variables de entorno de Linux distinguen entre mayúsculas y minúsculas. Si se desea ver el valor de una variable de entorno específica, podemos hacerlo pasando el nombre de esa variable como argumento al comando printenv.

La sintaxis básica para crear una variable de entorno se vería así:

export VAR="value"

Donde:

* export: el comando utilizado para crear la variable.
* VAR: el nombre de la variable.
* = indica que la siguiente sección es el valor.
* «value»: el valor real.

Para revertir el valor de una variable de entorno podemos usar el comando unset. La sintaxis del comando se ve de la siguiente manera:

unset VAR

Dónde:

* unset: el comando en sí
* VAR: la variable cuyo valor queremos revertir

Variables de entorno Linux locales y globales.

En términos de programación computacional, una variable global es aquella que se puede usar en cualquier parte del programa. Mientras que una variable local se encuentra definida en una función y solo puede usarse en esa función.

Las variables de entorno de Linux pueden ser globales o locales. Las variables de entorno global son visibles desde una sesión de shell y en cualquier proceso secundario que genera el shell. En cambio, las variables locales solo pueden estar disponibles en el shell en el que se crean.

Todas las variables de entorno del sistema utilizan letras mayúsculas para distinguirlas de las variables de entorno normales del usuario.

En el siguiente ejemplo se muestra como establecer una variable de entorno local de Linux. De modo que local\_var solo es visible en el shell actual:

local\_var = edward

echo $local\_var

Edward ßSalida

Podemos crear una variable de entorno global utilizando el comando export:

export Global\_var = Hello

bash

echo $Global\_var

Hello ßSalida

# **Desarrollo**

1.- Investiga el uso de los siguientes comandos y muestra su funcionamiento en una terminal:

|  |  |  |  |
| --- | --- | --- | --- |
| cal | clear | apt | rm |
| date | ifconfig | exit | mv |
| echo | df | ps | more |
| time | du | ps -fea | less |
| uname | pstree | man | mkdir |
| w | kill -l -9 | cat | pico |
| who | trap –l | fg | nano |
| bash | pwd | cd | vi |
| wc | su | ls |  |
| apt-get | sudo | ls -la |  |

* **cal**

Instrucción que muestra el mes en el que se encuentra el año y remarca el día en que se está, como se muestra en la imagen 6, sin embargo, cuenta con opciones de argumento, por ejemplo, si se pone *cal –y* aparece el calendario completo del año en curso como se muestra en la imagen 7.

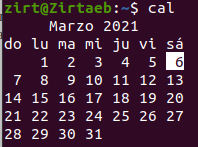


Imagen 6. Comando ‘cal’ sin argumentos

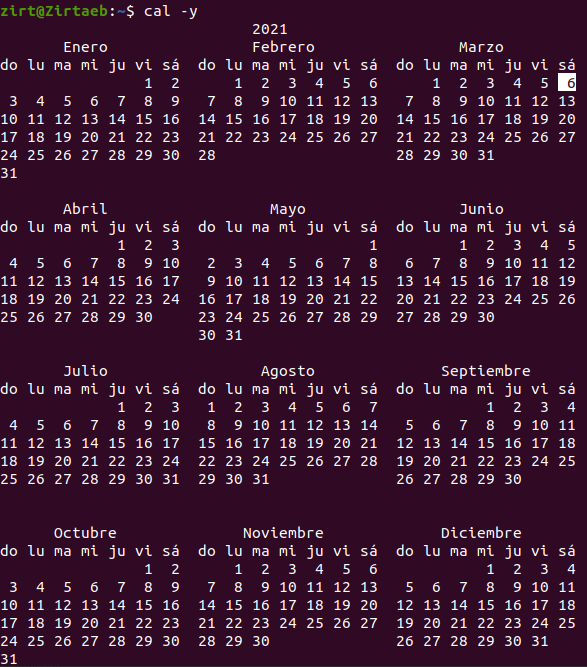


Imagen 7. Comando ‘cal’ con el argumento ‘y’

* **date**

Primero que nada esta instrucción sin ningún argumento muestra la fecha y la hora del momento en que se pone el comando como lo muestra la imagen 8, también este comando se le pueden poner argumentos para que cambie el formato de como muestra la fecha y la hora un ejemplo es como se muestra en la imagen 9, para saber que argumento usar, se pone *date - -help*, como lo vemos en la imagen 10, también con este comando vemos las secuencias que se interpretan para el formato como se ve en la imagen 11.



Imagen 8. Comando ‘date’ sin argumentos



Imagen 9. Comando ‘date’ con argumentos para cambiar formato de fecha y hora

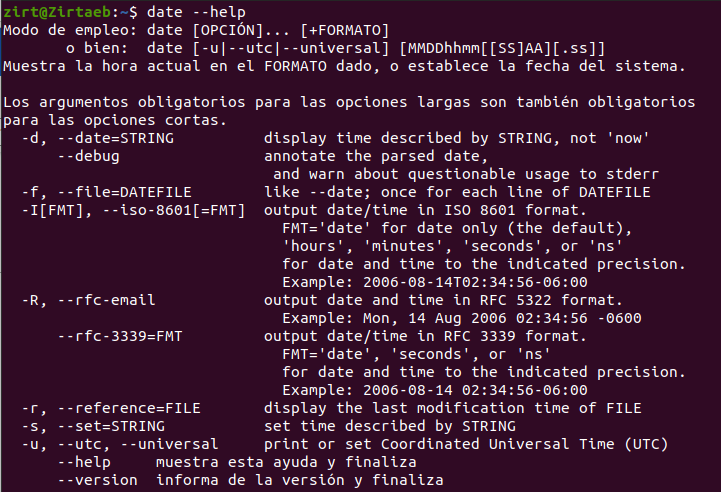


Imagen 10. Argumentos que pueden acompañar al comando ‘date’

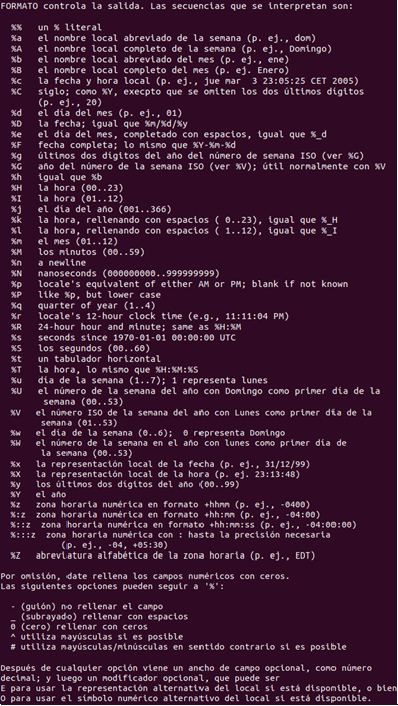


Imagen 11. Secuencias de ‘date’ para mostrar la salida en otro formato

* **echo**

Este imprime un texto en pantalla como se observa en la imagen 12, además que puede guardar lo que se escribe en una variable y también guarda el texto en un archivo, estos 3 últimos se muestran juntos en la imagen 13, donde primero se pone el texto en una variable, luego se guarda la variable en un archivo y por último se imprime el archivo con el comando *cat*. En la imagen 14, se puede ver el archivo de texto creado.



Imagen 12. Comando ‘echo’ con una cadena de caracteres

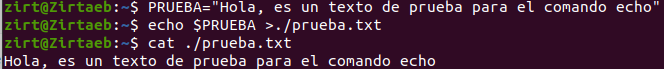


Imagen 13. Comando ‘echo’ con variable creada y guardado de la variable en un archivo

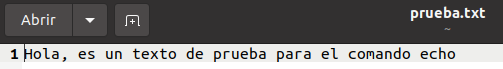


Imagen 14. Archivo guardado con ‘echo’

* **time**

Lo que hace este comando es que determina la duración de ejecución de un determinado proceso y pondrá en pantalla: *real* (tiempo transcurrido entre la ejecución y la finalización del proceso), *user* (tiempo de CPU gastado en código de modo usuario, fuera del núcleo, durante la ejecución del proceso) y *sys* (tiempo de CPU que transcurre en el núcleo al ejecutar el proceso), como a continuación se ve en la imagen 15. Cabe decir que también se realiza el proceso que indicamos después de time.

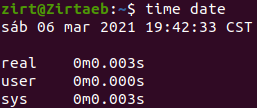


Imagen 15. Comando ‘time’

* **uname**

Este comando es la abreviatura de *Unix Name* y muestra cierta información del sistema, como el kernel. Primero, si se pone el comando sin argumentos solo nos devuelve el kernel como se observa en la imagen 16. Ahora si se pone el argumento *–a* se podrá observar toda la información que ofrece el comando como se muestra en la imagen 17. Los argumentos que se pueden poner se pueden observar si se escribe *uname - -help* como se muestra en la imagen 18.



Imagen 16. Comando ‘uname’ sin argumentos



Imagen 17. Comando ‘uname’ con el argumento ‘a’

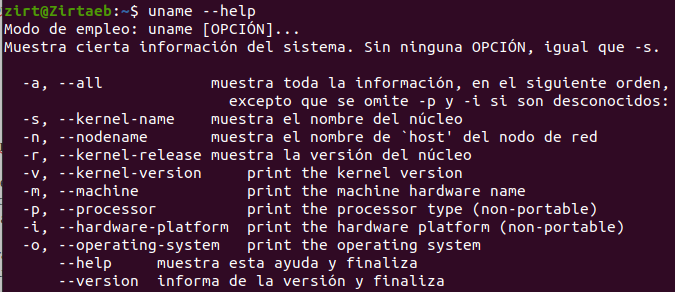


Imagen 18. Argumentos que pueden acompañar al comando ‘uname’

* **w**

Este comando muestra información sobre los usuarios actualmente conectados y lo que está haciendo cada usuario, como se observa en la imagen 19, además que se pueden poner varios argumentos los cuales podemos ver poniendo *w - -help* como se muestra en la imagen 20.

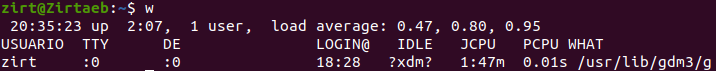


Imagen 19. Comando ‘w’ sin argumentos

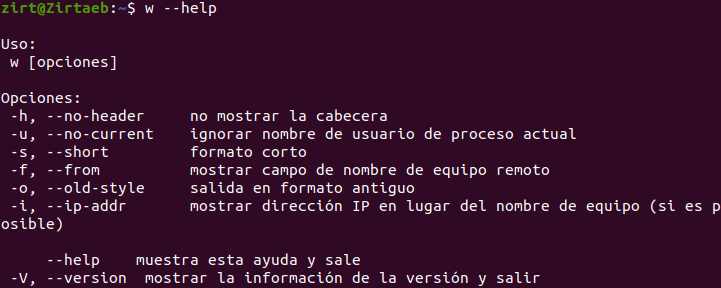


Imagen 20. Argumentos que pueden acompañar al comando ‘w’

* **who**

Este comando muestra información sobre los usuarios que están activos, como su hora de arranque del sistema, como lo vemos en la imagen 21, claro que lo que vemos en la imagen es su uso sin argumentos, los argumentos que se pueden ocupar son varios y se pueden ver si se pone *who - -help* como lo vemos en la imagen 22.



Imagen 21. Comando ‘who’ sin argumentos

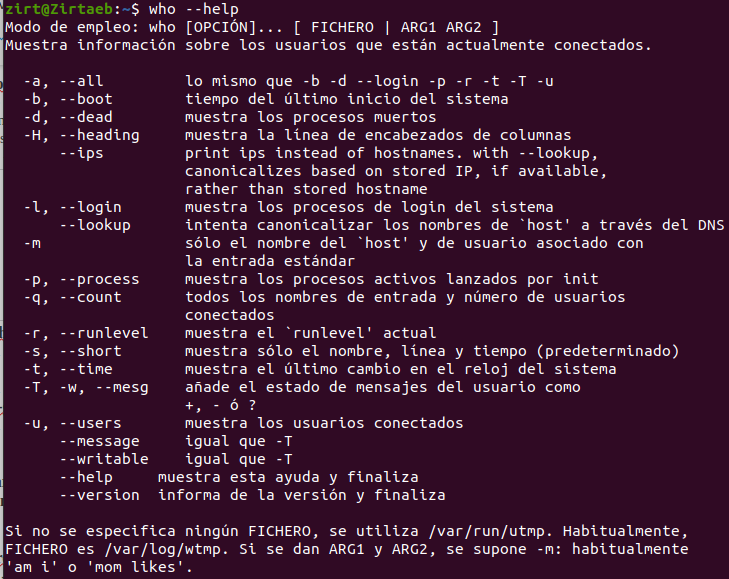


Imagen 22. Argumentos que pueden acompañar al comando ‘who’

* **bash**

El Proyecto GNU desarrolló un shell de software libre para ser parte de su sistema operativo y lo llamó "Bourne Again Shell" (bash). Cuando se inicia una ventana de terminal en Linux, el sistema inicia el programa de shell. Las shells ofrecen una forma estándar de extender el entorno de línea de comando.

Al escribir *bash* sin argumentos, aparentemente no pasa nada, como se ve en la imagen 23, esto es porque, como se menciona arriba, bash ya se está ocupando desde que se abre la terminal.

Por otro lado, si ponemos bash con algún argumento por ejemplo *bash –version*, se puede ver la versión como lo muestra la imagen 24, tiene otros argumentos los cuales se pueden ver al poner *bash - -help*, como se ve en la imagen 25.



Imagen 23. Comando ‘bash’ sin argumentos

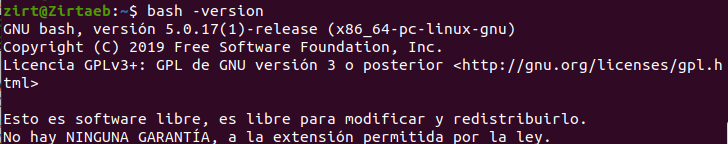


Imagen 24. Versión de bash

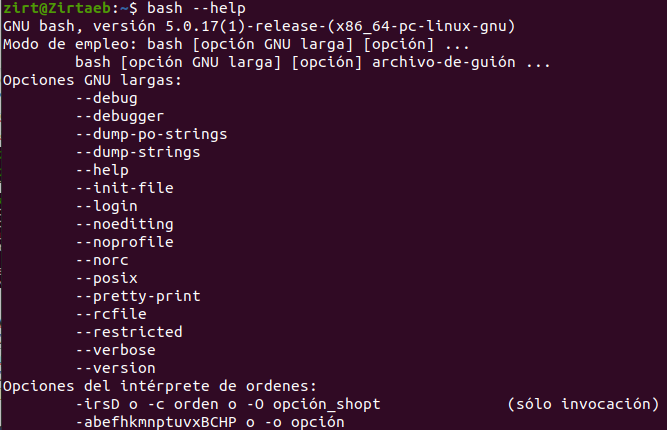


Imagen 25. Argumentos que pueden acompañar al comando ‘bash’

* **wc**

Es la abreviación de word count (conteo de palabras), lo que hace el comando es que de un archivo devuelve el número de líneas, el número de palabras, entre otros. En la imagen 26, se puede ver que devuelve el conteo de líneas, palabras y bytes del archivo creado anteriormente en el comando *echo*, además se puede ver específicamente algo, por ejemplo, solo el conteo de líneas, ocupando argumentos los cuales se pueden conocer poniendo *wc - -help* como se muestra en la imagen 27.



Imagen 26. Comando ‘wc’



Imagen 27. Argumentos que pueden acompañar al comando ‘wc’

* **apt-get**

Este comando obtiene paquetes e información de estos, los cuales se pueden actualizar, instalar o eliminar, si se escribe solo nos muestra la información de uso y los argumentos que lo pueden acompañar como lo muestra la imagen 28, esta información es la misma que mostraría si se pusiera *apt-get - -help*.

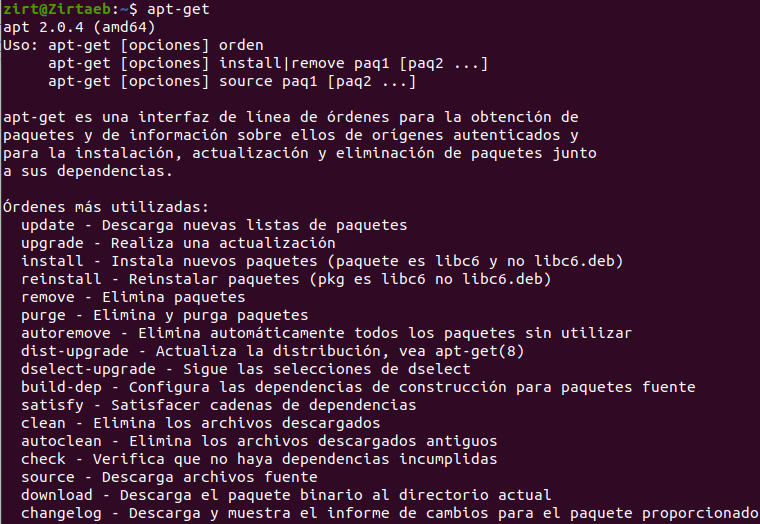


Imagen 28. Comando ‘apt-get’

* **clear**

Este comando borra la información que se tiene en la pantalla de la terminal, por ejemplo, en la imagen 29 utilizamos el comando cal –y:



Imagen 29. Antes de usar el comando clear.

El comando clear está está escrito, si damos enter la pantalla se limpiará.



Imagen 30. Después de usar el comando clear.

* **ifconfig**

Permite configurar o desplegar parámetros de las interfaces de red residentes en el núcleo, como la dirección IP, la máscara de red. Utilizamos este comando cuando necesitemos consultar información relacionada a la configuración de red de nuestro equipo como se muestra en la imagen 31.

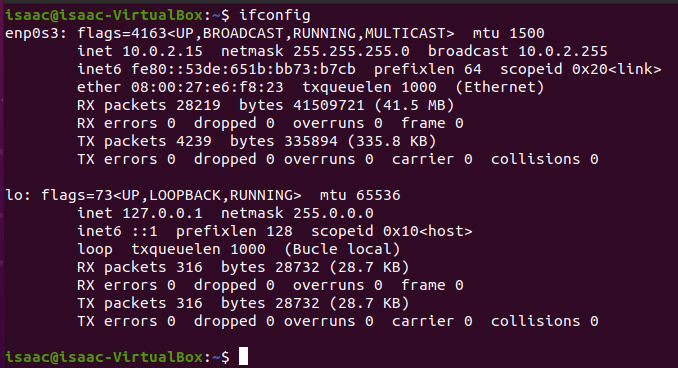


Imagen 31. Comando ‘ifconfig’

* **df**

Muestra el espacio en disco utilizado por el sistema de ficheros, como se puede observar en la imagen 32.

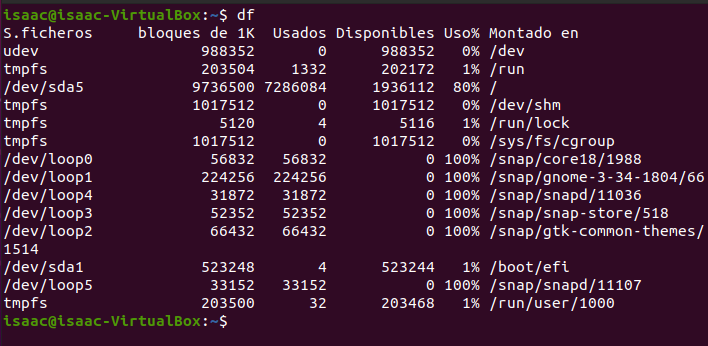


Imagen 32. Comando ‘df’

* **du**

El comando du (disk usage que significa uso de disco) se usa para estimar el uso de espacio en disco duro de un archivo, un directorio en particular o de archivos en un sistema de archivos. Se ve su uso en la imagen 33.

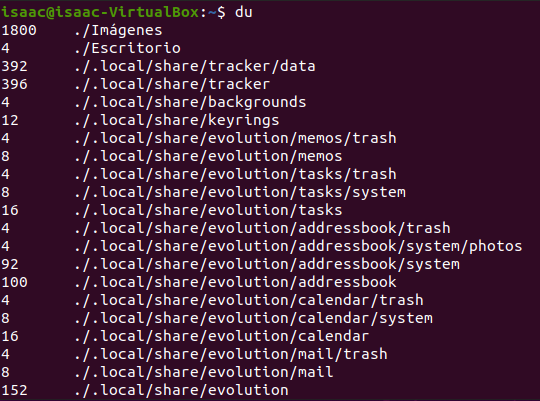


Imagen 33. Comando ‘du’

Si queremos ver específicamente el peso de un archivo o carpeta de la forma más fácil, ejecutamos:

$ du -bsh Vídeos/, como se muestra en la imagen 34.

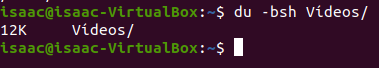


Imagen 34. Uso del comando ‘du -bsh’

Si queremos ver cuáles son los directorios más pesados que tenemos en nuestro /home organizados de mayor a menor: $ du -sm \*, como se muestra en la imagen 35.

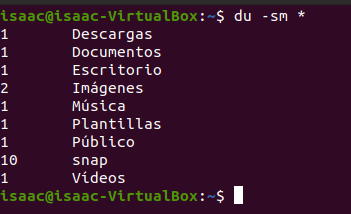


Imagen 35. Comando ‘du -sm \*’

Si sólo quisiéramos ver cuáles son, por ejemplo, los 5 directorios más pesados en nuestro /home podríamos usar du con una serie de comandos extras, por ejemplo:

$ du -sm \* | sort -nr | head –5, que es mostrado en la imagen 36.

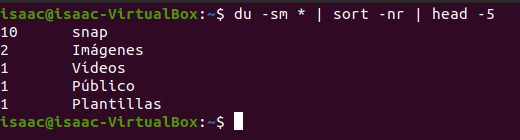


Imagen 36. Comando du con X directorios más pesados.

El siguiente comando muestra lo mismo que el comando anterior pero representados en MB (véase la imagen 37):

$ du -hs \* | sort -nr | head -5

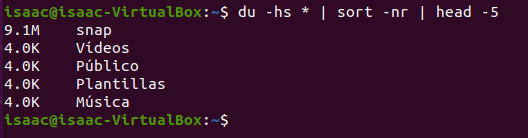


Imagen 37. Comando du con X directorios más pesados representados en MB.

* **pstree**

Este comando nos permite ver el árbol de procesos activos, como se muestra en la imagen 38.

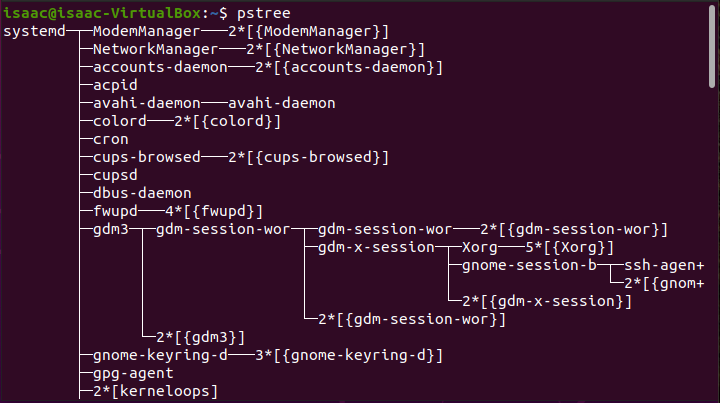


Imagen 38. Comando ‘ps’ sin argumentos.

Con el parámetro “**-a**”, nos muestra la línea de comandos utilizada. Por ejemplo, si el comando utiliza la ruta a un fichero de configuración.

Por defecto se inhabilita la visualización en el árbol de los nombres repetidos. Para evitar esto, podemos utilizar el parámetro “**-c**”

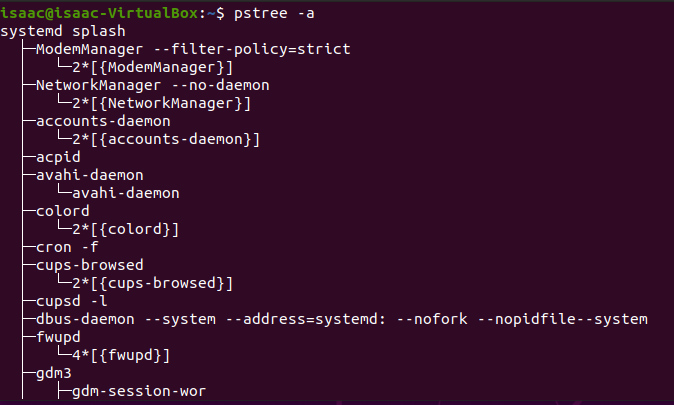


Imagen 39. Comando ‘ps’ con argumento –a.

Utilizando el parámetro “**-g**”, veremos los procesos agrupados por grupo.

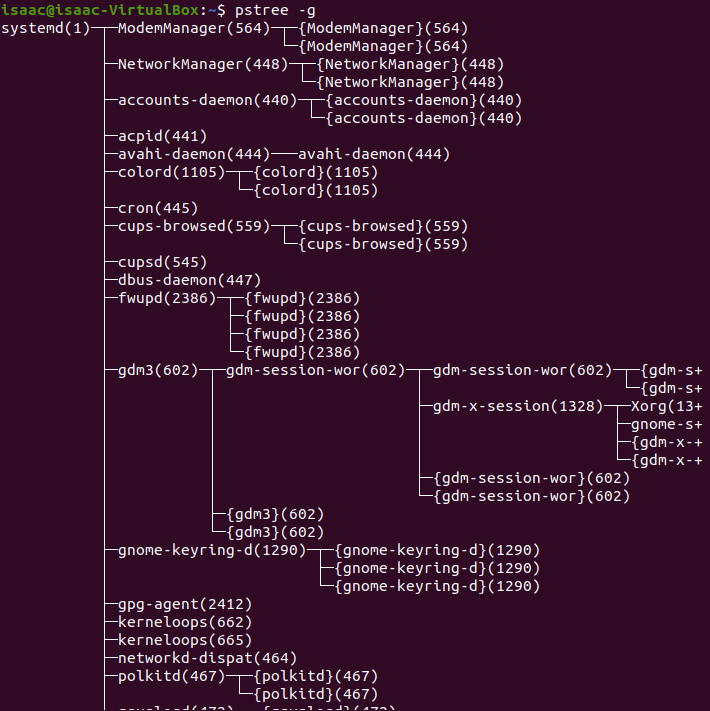


Imagen 40. Comando ‘ps’ con argumento –g.

* **kill -l -9**

El comando kill mata un proceso usando su PID (identificador de proceso, es un numero entero usado por el kernel del sistema operativo para identificar un proceso de manera unívoca).

Para matar un proceso utilizando el comando kill primero debemos conocer el identificador del proceso que queremos aniquilar, para ello hacemos uso del comando **ps -fea | grep** seguido del nombre del proceso, por ejemplo: **ps -fea |grep htop**

donde nos va a arrojar información del proceso que está corriendo con el nombre htop y el identificador del mismo.

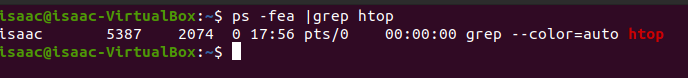


Imagen 41. Conociendo el PID a aniquilar.

Una vez conozcamos el identificador del proceso debemos ejecutar el comando **kill** seguido del parámetro correspondiente, en este caso debemos utilizar el parámetro -9 que se encarga de matar de manera segura el proceso. Por ejemplo: **kill -9**



Imagen 42. Comando ‘kill -9’

Para eliminar todos los procesos para los que el usuario tiene permiso, excepto el proceso raíz (PID 1) y el proceso de eliminación en sí utilizamos **kill -1**

* **trap –l**

Trap permite capturar señales y ejecutar código cuando ocurren. Las señales son notificaciones asincrónicas que se envían a un script cuando ocurren ciertos eventos. La mayoría de esas notificaciones son para eventos que se espera que nunca sucedan, como un acceso a memoria no válido o una llamada incorrecta al sistema. Sin embargo, hay uno o dos eventos con los que posiblemente desee lidiar.

La sintaxis es la siguiente:

trap [-lp] [[arg] signal\_spec ...]

trap [action] [signal]

Ahora mostramos la función de sus parámetros:

-l Se utiliza para mostrar la lista de todos los nombres de señales con el número correspondiente.

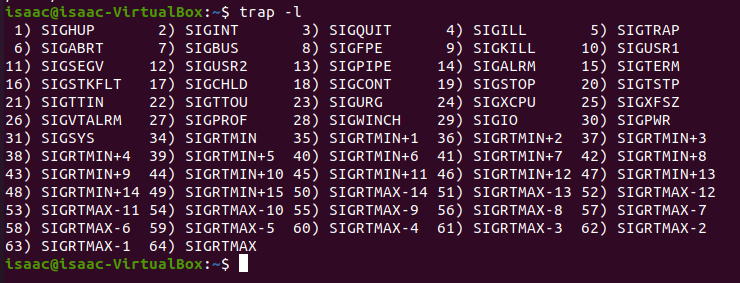


Imagen 43. Comando ‘trap -l’

–pag se utiliza para mostrar el comando de señal o el comando de trap para signal\_spec.

arg se utiliza para ejecutar un comando cuando el shell recibe la (s) señal (es).

signal\_spec contiene el nombre de la señal o el número de la señal.

* **pwd**

El comando pwd donde sus siglas provienen del inglés ***p****rint* ***w****orking* ***d****irectory*, cuya traducción sería imprimir directorio de trabajo*,* se utiliza para imprimir el nombre del directorio actual en una sesión de comandos. Si en el intérprete de comandos no lo está mostrando ya, el usuario puede invocar este comando para averiguar cuál es su ubicación en el árbol de directorios del [sistema de ficheros](https://denovatoanovato.net/sistema-de-archivos-fhs/).

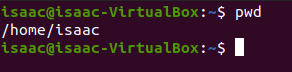


Imagen 44. Comando ‘pwd’.

* **su**

Este comando permite usar el [intérprete de comandos](https://es.wikipedia.org/wiki/Int%C3%A9rprete_de_comandos) de otro usuario sin necesidad de cerrar la sesión. Comúnmente se usa para obtener permisos de [root](https://es.wikipedia.org/wiki/Root) (superusuario) para operaciones administrativas, sin tener que salir y reentrar al sistema.

Si acompañamos el comando ***su*** con el nombre de usuario al que nos queremos conectar, Linux nos solicitará la contraseña y accederemos directamente a dicha cuenta. En caso de querer regresar al usuario original, escribiremos el comando ***exit***.

Ejemplo:

su carlos

Después de haber insertado la contraseña, nos encontraremos bajo el usuario "carlos", seguiremos en la misma carpeta de trabajo que antes y nos beneficiaremos de sus prestaciones (shell, permisos...).

Para acceder al **superusuario o root**, únicamente tendremos que escribir:

su

De nuevo, se nos solicitará la contraseña y accederemos como **administrador**.

Para iniciarun nuevo shell de conexión**:**

El guión ("**-**") fuerza el inicio de un nuevo shell de conexión, con las preferencias por defecto del usuario **"carlos"**.  
  
Ejemplo:

su – carlos

Para volver al shell anterior

Escribir el comando "**exit**" o la combinación de las siguientes teclas "**Ctrl+D**" para cerrar el shell y volver al anterior.

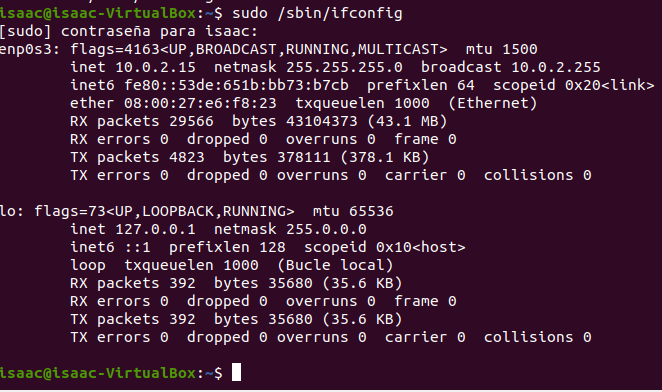
* **sudo**

El comando sudo permite a los usuarios ejecutar programas con los privilegios de seguridad de otro usuario (normalmente el usuario root) de manera segura, convirtiéndose así temporalmente en súper usuario.

Si un usuario normal desea ejecutar un comando de root (o de cualquier otro usuario), Sudo verifica en su lista de permisos y si está permitido la ejecución de ese comando para ese usuario, entonces Sudo se encarga de ejecutarlo

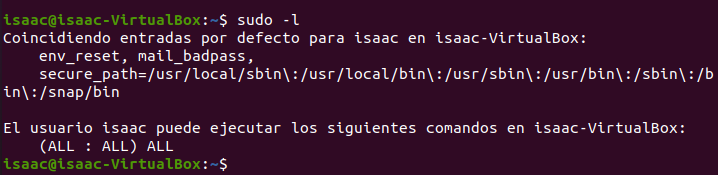
Pasos para utilizar sudo:

1.- Sudo lo ejecuta un usuario normal, al que se supone tiene permisos para ejecutar cierto comando. Entonces, sudo requiere que los usuarios se autentifiquen a sí mismos a través de su contraseña para permitirles la ejecución del comando, usamos sudo /sbin/ifconfig.

Imagen 45. Comando ’sudo’.

Después de hacer lo anterior tendrás 5 minutos para volver a usar el mismo comando u otros a los que tuvieras derecho, sin necesidad de ingresar la contraseña de nuevo.

2.- ¿Qué comandos son los que puedo utilizar?, pues la opción -l es la indicada para eso:

Imagen 46. Comando ’sudo -l’

En el caso anterior se ejecutó un comando de root, pero no tiene que ser así, también es posible ejecutar comandos de otros usuarios del sistema indicando la opción -u:

* **apt**

APT (Advanced Package Tool) es una herramienta de línea de comandos que interactúa con el sistema de empaquetado y actualmente tenemos comandos dpkg su administración. Con apt podemos realizar las operaciones más comunes sobre paquetes y/o programas desde la linea de comandos en Linux.

Alguans de las operaciones básicas son:

* + list - lista los paquetes según los nombres
  + show - muestra detalles del paquete
  + install - instala paquetes
  + remove - elimina paquetes
  + update - actualiza la lista de paquetes disponibles

En la imagen 47, se muestra un ejemplo de cómo usar apt para actualizar la base de datos de los paquetes.

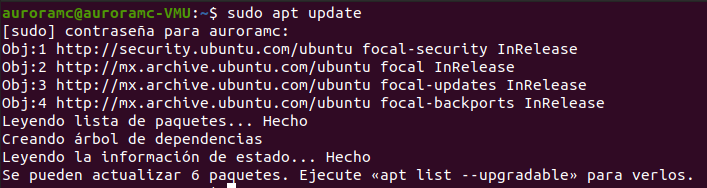


Imagen 47. Uso de apt en la terminal de Ubuntu.

* **exit**

Exit es un comando básico en Linux que cierra las ventanas, las conexiones remotas o la pantalla del terminal.

* **ps**

El comando ps muestra por pantalla un listado de los procesos que se están ejecutando en el sistema. Si no añadimos ningún parámetro, ps mostrará los procesos del usuario con el que estamos logueados.

Por otra parte, los parámetros más básicos a conocer son los siguientes:

* aux: Lista los procesos de todos los usuarios con información añadida (destacamos más abajo).
* a: Lista los procesos de todos los usuarios.
* u: Lista información del proceso como por ejemplo el usuario que lo está corriendo, etc.
* x: Lista procesos de todas las terminales y usuarios
* l: Muestra información que incluye el UID y el valor «nice«.

Ejemplo de uso del comando ps:

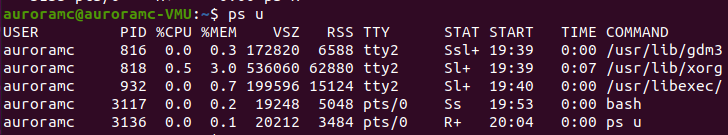


Imagen 48. Uso de ps con el parámetro “u” en la terminal de Ubuntu.

* **ps -fea**

Lista los procesos activos detalladamente.

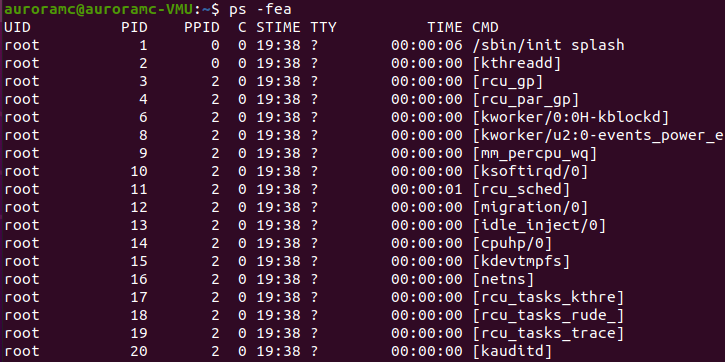


Imagen 49. Uso de ps con el parámetro “-fea” en la terminal de Ubuntu.

* **man**

Con man se abren las páginas del manual (man pages) de tu distribución de Linux en la consola. Si, por ejemplo, quieres abrir el manual para un comando específico, se combina man con el nombre del comando: “man clear”.

En ambos casos se abre la página del manual por el comando clear. Para cerrarlo y volver a la consola utiliza la tecla [Q].

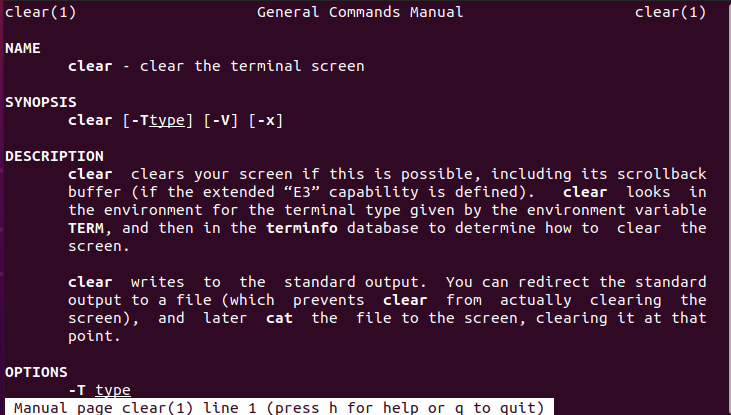


Imagen 50. Uso de man con el parámetro “clear” en la terminal de Ubuntu.

* **cat**

Deriva su nombre de la palabra concatenar y te permite crear, fusionar o imprimir archivos en la pantalla de salida estándar o en otro archivo y mucho más. El comando toma un nombre de archivo como argumento junto con opciones para especificar operaciones particulares.

cat [OPTION] [FILE]

Ejemplo de uso:



Imagen 51. Uso de cat para crear un archive .txt en la terminal de Ubuntu.

* **fg**

El comando fg (foreground o primer plano) traerá a primer plano un trabajo que está ejecutándose en segundo plano. También se puede usar para reanudar en primer plano un trabajo que está suspendido o detenido.

* fg %1 para poner en primer plano la tarea cuyo número de tarea es 1 (du -ah /).
* fg %du para poner en primer plano la tarea cuyo nombre empiece por du
* fg %?ah para poner en primer plano la tarea que contenga la cadena ah
* fg o fg %% para poner en primer plano la última tarea (la cual es du -ah /).

Una vez la tarea esté en primer plano la suspenderemos presionando la combinación de teclas Ctrl+Z.

En el siguiente ejemplo, primero empezamos un trabajo en segundo plano para poder probar el comando fg, entonces podrá pasar ese trabajo a primer plano y luego lo suspenderemos.

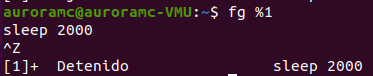


Imagen 52. Uso de fg en la terminal de Ubuntu.

* **cd**

El comando cd es la abreviatura de change directory y se utiliza para navegar por el directorio. La sintaxis de esta orden sigue el esquema:

cd [OPCIÓN] DIRECTORIO

Si no se indica ningún directorio concreto, cd cambia automáticamente al directorio principal del usuario. Si se acompaña de un guion (-), cd abre el directorio precedente.



Imagen 53. Uso de cd en la terminal de Ubuntu.

* **ls**

La orden ls equivale a list y se utiliza para mostrar el contenido de un fichero (los nombres de todos sus archivos y carpetas). Si no se añade a ls ningún directorio, el comando enumera el contenido del directorio en curso. Con ayuda de diferentes opciones se puede definir qué información se ha de mostrar y cómo.

En el siguiente ejemplo, se utiliza el parámetro -l, el cual permite ver información adicional como permisos del fichero, el número de enlace, nombre del propietario, nombre del grupo al que pertenece, tamaño en bytes, una marca de tiempo y nombre del fichero, como se muestra en la imagen 54.

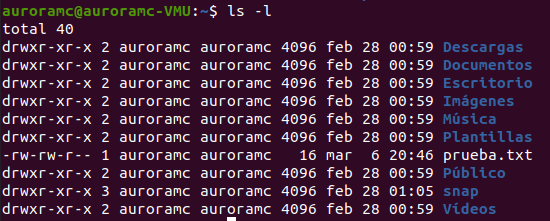


Imagen 54. Uso de ls con el parámetro -l en la terminal de Ubuntu.

* **ls -la**

Este comando es muy parecido al comando de ls, simplemente este recibe dos parámetros: l y a, esto quiere decir que va mostrar todos los elementos dentro del directorio, hasta los elementos que comienzan con “.” y l significa que lo mostrará en forma de lista en una manera más detallada donde nos muestra los permisos, el tamaño, los usuarios, la fecha de modificación, si nombre, etc.

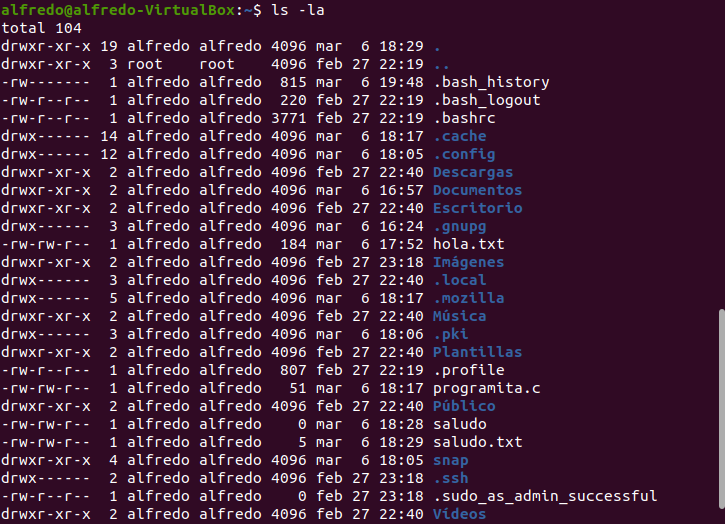
A continuación, veremos un ejemplo.

Imagen 55. Ejemplo del uso del comando ls -la.

* **rm**

Es una abreviación para “remove”. Este comando está hecho para eliminar ficheros, y su formato es rm [opción]…[fichero]…

Entre sus opciones temenos:

* -f, --force descarta los ficheros u opciones que no existan sin preguntar.
* -i pregunta antes de cada borrado.
* -I pregunta antes de borrar más de 3 ficheros o cuando se borra recursivamente.
* –interactive (=CUANDO), pregunta dependiendo la opción que ingresemos: never (nunca), once (una vez, -I), o always (siempre, -i), cuando no se especifica lo toma como always.

A continuación, veremos el ejemplo poniendo always.

Imagen 56. Ejemplo usando rm –interactive.

* –one-file-system, al eliminar una jerarquía recursivamente, omite cualquier directorio que está en un directorio diferente del que corresponde al argumento de la línea de comandos.
* –no-preserve-root, no trata a “/” de manera especial.
* –preserve-root, no eliminar a “/” o lo trata como raíz.
* -r, -R, --recursive, elimina directorios y su contenido recursivamente.

A continuación, veremos un ejemplo de este comando.

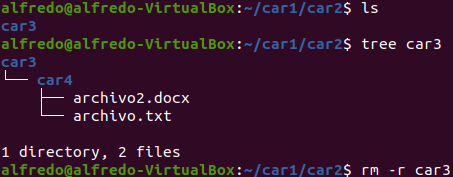


Imagen 57. Ejemplo del uso de rm -r.

Nos ayudamos del comando tree para ver de forma mejor cómo estaban los directorios, después de usar el comando se eliminó car3 y todo lo que había dentro.

* -d, --dir, elimina los directorios vacíos sin usar -r y sus versiones.
* -v, --verbose, explica que se está haciendo.
* –help, muestra la ayuda.
* –version, muestra la versión.
* Para eliminar un archivo que empiece con “-“ ejecutamos el comando rm -- -archivo.txt
* **mv**

Es abreviación de move. Su modo de uso es: mv [opción]… origen destino, mv [opción]… origen… directorio, mv [opción]… -t directorio origen. La primera forma renombra, la segunda forma mueve.

Sus opciones son:

* –backup[=CONTROL], crea una copia de seguridad antes de borrar
* -b como –backup pero sin argumento.
* -f, --force, no pregunta antes de sobrescribir.
* -i, --interactive, pide confirmación antes de sobrescribir
* –strip-trailing-slashes, elimina todas las barras finales de cada argumento origen.
* -S, --suffix=SUFIJO, reemplaza el sufijo del archivo destino con el dado.
* -t, --tarjet-directory=DIRECTORIO, mueve todos los recursos de la carpeta origen al destino.
* -T, --no-tarjet-directory, trata el destino como archivo y no como directorio.
* -u, --update, va moverse solamente si el directorio destino es más viejo que el recurso o aún no ha sido creado.
* -v, --verbose, da detalles del movimiento que se está haciendo.
* –version, muestra la versión del comando.
* –help, muestra ayuda sobre el comando.

Ahora veremos el ejemplo moviendo un directorio. Primero mostramos cómo se encuentran los directorios.

Imagen 58. Directorios antes de mover.

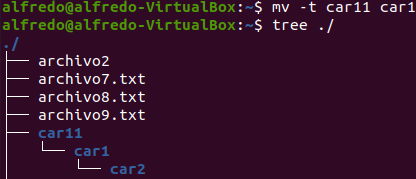
Ejecutamos el comando y vemos cómo quedan los directorios ahora, movemos la car1 dentro de car11.

Imagen 59. Directorios después de moverlos.

* **more**

Sirve para visualizar y filtrar la información resultante de un comando. Y tiene las siguientes opciones:

* -d, muestra ayuda cuando es presionada una tecla incorrectamente.
* -f, cuenta las líneas lógicas en vez de las mostradas en pantalla.
* -l, suprime la pausa posterior al avance de la página.
* -c, no desplaza, muestra texto y limpia los finales de línea.
* -p, no desplaza, limpia la pantalla y muestra texto
* -s, junta varias líneas en blanco en solo una.
* -u, suprime subrayados.
* -<numero>, número de líneas por pantalla complete.
* +<numero>, muestra el comienzo del fichero desde el número de línea.
* +/<cadena>, muestra el comienzo del fichero desde la coincidencia de la cadena de búsqueda.
* –help, muestra ayuda.
* -v, --verbose, muestra la versión del comando.
* **less**

Es un comando similar a more, pero en este caso nos encontramos con más características, no necesita leer todo el archivo antes de comenzar, eso hace que con archivos de gran magnitud comience mucho más rápido, los comandos son basados en more y en vi, comandos pueden ser procedidos por un decimal llamado N, este número es usado en algunos comandos.

* **mkdir**

Es una abreviatura para make directory. Crea directorios en caso de que no existan, su formato es así: mkdir [opción]… directorio…

Tiene las siguientes opciones:

* -m, --mode=MODE, establece los permisos como se hace en chmod.
* -p, --parents, no hay error en caso de que existan, y si no existen los crea.

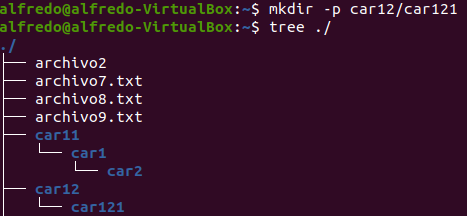
Veremos un ejemplo de este comando con ese argumento a continuación:

Imagen 60. Ejemplo de mkdir -p.

* -v, --verbose, muestra un mensaje por cada directorio creado.
* -Z=context, --context=CONTEXT, si se está usando SELinux está opción pone a el contexto de cada directorio creado.
* –help, muestra ayuda.
* –version, muestra la versión del comando.
* **pico/nano**

Es un editor de texto dentro de la terminar de las distribuciones modernas de Linux, tiene una manera muy sencilla de usarse, la manera más sencilla es ejecutar del comando pico seguido del nombre del archivo y así abrimos el archivo con el editor, nos muestra una pantalla donde podemos editar el texto del archivo además de algunas opciones para administrar el archivo. A continuación, se muestra.

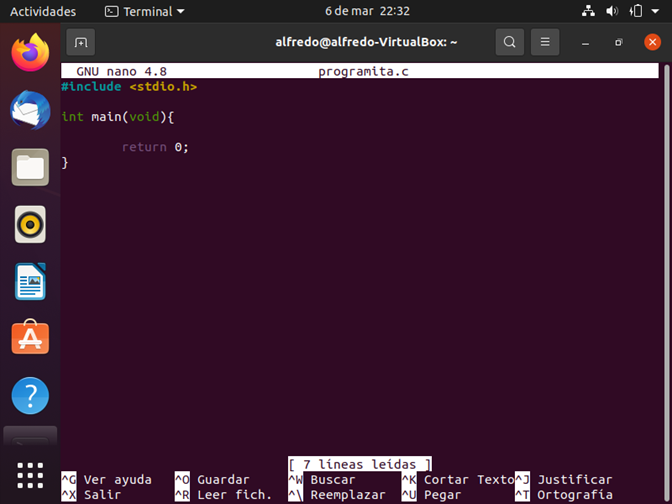


Imagen 61. Visualización del editor de nano(pico).

Dentro del editor tenemos varios comandos, para ejecutarlos simplemente seguimos la combinación de teclas de cada comando siendo el carácter “**^**” la tecla de ctrl. Nano es un editor muy completo por lo tanto tiene cientos de comandos y opciones.

* **Vi**

Este de igual forma es un editor de texto igual de completo que nano y tiene bastantes funciones, cientos de comandos a usar, para abrirlo basta con escribir la palabra vi seguido del nombre del archivo a editar.

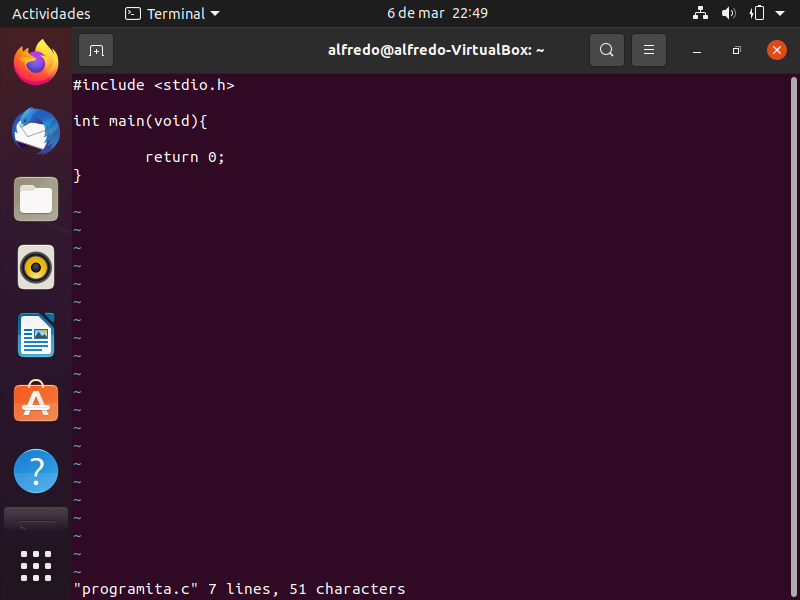
Ahora muestro un ejemplo del uso de Vi.

Imagen 62. Visualización de Vi.

2.- Escribe y explica al menos tres ejemplos de direccionamiento absoluto y tres ejemplos de direccionamiento relativo. Escribe y explica tres ejemplos de borrado de archivos utilizando el comando rm y los comodines “\*” y “?”.

Primero mostraré los ejemplos de direccionamiento absoluto, utilizamos el comando pwd para mostrar donde nos encontramos inicialmente:

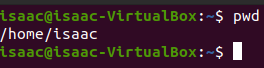


Imagen 63. Uso del comando pwd

Ahora ejecutamos cd /home/isaac/Vídeos/ESCOM/Sistemas\_Operativos para ir a el directorio Sistemas\_Operativos, y luego ejecutamos pwd para mostrar donde nos encontramos:

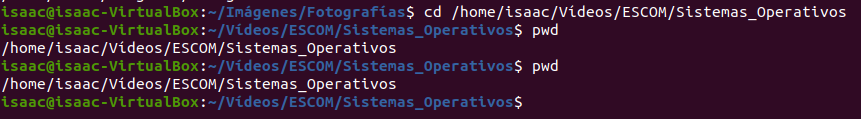


Imagen 64. Ejemplo 1 de direccionamiento absoluto.

Es importante saber que los directorios no deben tener espacios, ya que si los tienen no podremos acceder a esos directorios. Además, debemos escribir los directorios completamente igual, con mayúsculas, minúsculas, acentos, etc.

Para el siguiente ejemplo usamos cd /home/isaac/Documentos/ para ir al directorio Documentos, igualmente ejecutamos el comando pwd:

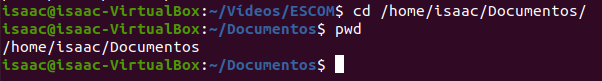


Imagen 65. Ejemplo 2 de direccionamiento absoluto.

Ahora estamos en el directorio Documentos utilizando direccionamiento absoluto.

Ahora, si queremos ir al directorio Fotografías usamos cd /home/isaac/Imágenes/Fotografías/

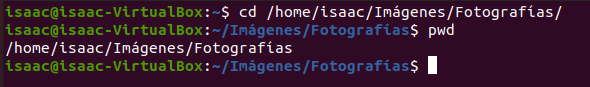


Imagen 66. Ejemplo 3 de direccionamiento absoluto.

Seguidamente, presentaré los ejemplos de direccionamiento relativo.

Recordando del ejemplo anterior que nos encontrábamos en el directorio Fotografías, si ejecutamos cd../.., vamos a el directorio isaac ya que fuimos al directorio padre del padre.

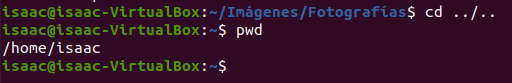


Imagen 67. Ejemplo 1 de direccionamiento relativo.

Pero si nos encontráramos en el directorio Sistemas\_Operativos (que lo vimos anteriormente) y ejecutamos cd ../.. veamos a donde nos dirigimos usando el comando pwd:

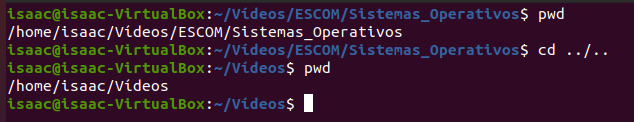


Imagen 68. Ejemplo 2 de direccionamiento relativo.

Como vemos, nos dirigimos al directorio Vídeos, a pesar de que usamos exactamente cd ../.., queriendo decir que el direccionamiento relativo importa en donde estemos mientras que el direccionamiento absoluto no importa en que directorio nos encontremos.

Para dar otro ejemplo, si nos encontráramos en el directorio ESCOM y usáramos cd ../.., nos mandaría a home:

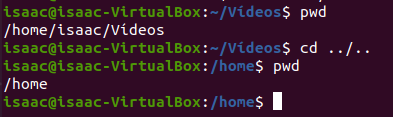


Imagen 69. Ejemplo 3 de direccionamiento relativo.

A continuación, los ejemplos de borrado de archivos usando rm y el comodín “\*”.

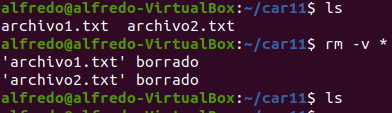
* Vamos a eliminar todos los archivos que se encuentran en una carpeta usando el comodín “\*”

Imagen 70. Ejemplo de comando rm usando \* para eliminar todos los archivos.

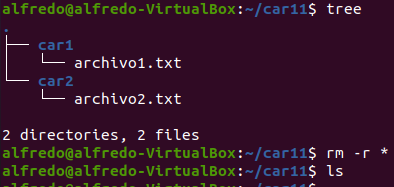
* Vamos a eliminar todas las carpetas que se encuentran en otra carpeta usando el comodín “\*” y usando -r.

Imagen 71. Ejemplo de comando rm usando \* para eliminar todas las carpetas.

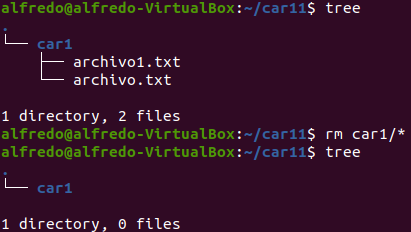
* Ahora bien, vamos a eliminar todos los archivos de una carpeta hija.

Imagen 72. Ejemplo de comando rm usando \* para eliminar archivos de una carpeta hija.

Por último, los ejemplos de borrado de archivos usando rm y el comodín “?”.

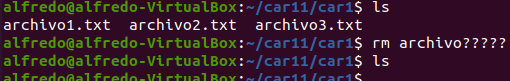
* Vamos a eliminar todos los archivos que comiencen con “archivo”, usando 5 comodines “?” al final.

Imagen 73. Ejemplo de comando rm usando “?” al final.

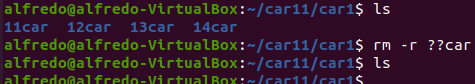
* Vamos a eliminar todas las carpetas que empiecen con dos números.

Imagen 74. Ejemplo de comando rm usando “?” para eliminar todas las carpetas.

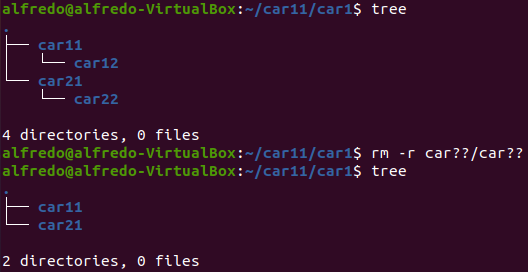
* Ahora bien, vamos a eliminar las carpetas que terminen con dos números que estén dentro de carpetas que también terminen con dos números.

Imagen 75. Ejemplo de comando rm usando “?” para eliminar carpetas hijas con el mismo formato que las carpetas madre.

3.- Escribe y explica dos ejemplos de redireccionamiento utilizando los comandos > y >>.

* **> |** Redirecciona hacía un archivo. Lo crea si no existe, si existe lo sobrescribe.

Ej. La salida del comando se envía a un archivo en vez de la terminal. El comando ls -l devuelve el contenido del fichero con información adicional, entonces al aplicar el redireccionamiento a lista.txt, esta información se escribirá en el archivo y se guardará.

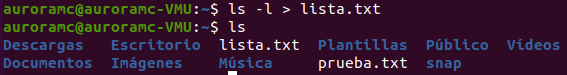


Imagen 76. Uso de ls con el parámetro -l y con un redireccionamiento a un archivo.

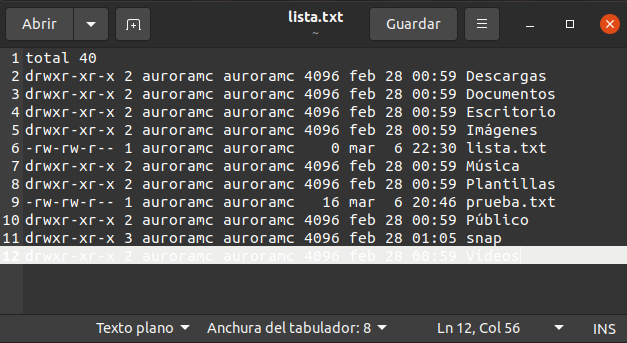


Imagen 77. Captura del archivo lista.txt

* >> | Redirecciona hacía un archivo. Lo crea si no existe, si existe concatena la salida al final de este.

Ej. El comando ps u devuelve información de los procesos que se están ejecutando y al hacer el redireccionamiento con >>, esta se concatena al archivo lista.txt que habíamos creado antes.

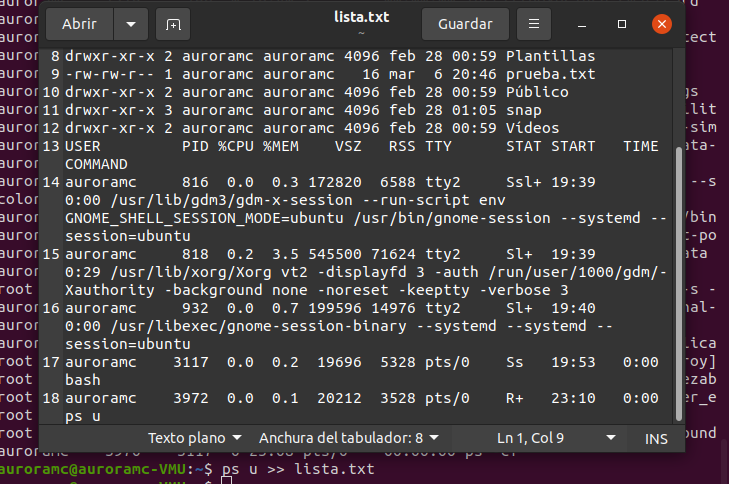
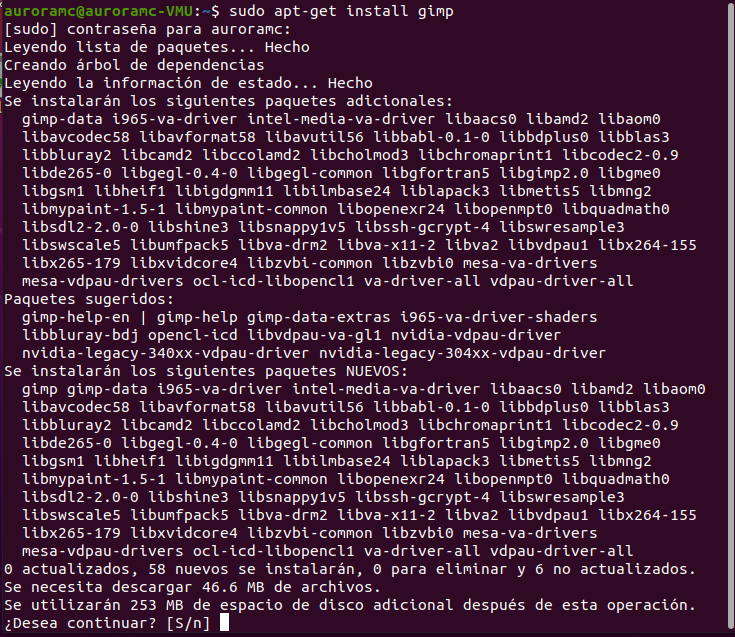


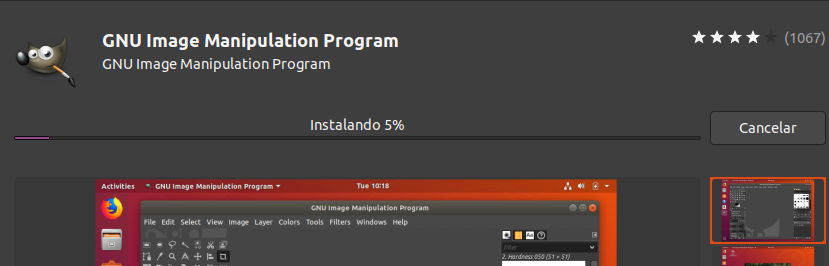
Imagen 78. Captura del archivo lista.txt actualizado.

4.- Instale algún software en línea de comandos (por ejemplo, instale el paquete gimp o konsole). Instale algún otro software usando el entorno gráfico.

* Instalación del paquete gimp por medio de una línea de comando y esperamos a que termine la descarga e instalación del mismo.

  
Imagen 79. Proceso de instalación del paquete.

* Para instalar un software usando el entorno gráfico, entramos al centro de software de Ubuntu, que es una tienda de software y buscamos la aplicación que queramos instalar e intuitivamente es fácil, pulsamos el botón de instalar y esperamos a que se concluya la descarga e instalación.

  
Imagen 80. Instalación de aplicación mediante la tienda de Software de Ubuntu.

5.- Escriba al menos cinco ejemplos utilizando la jerarquía en los directorios para mover y copiar archivos y directorios desde la terminal.

* Para mover un archivo de directorio utilizaremos el comando mv. Nos posicionaremos en el directorio donde está el archivo que queremos mover y después ejecutaremos el comando, el cuál sería el siguiente:

mv prueba.txt /home/auroramc/Documentos

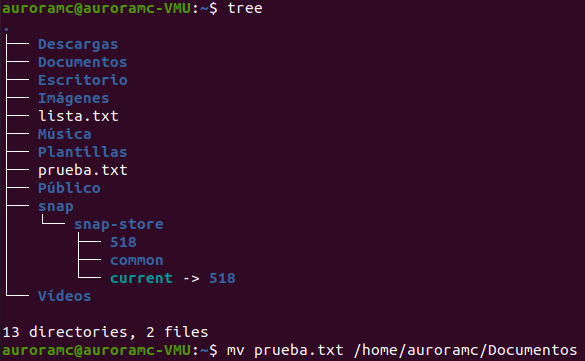


Imagen 81. Uso del comando mv para mover un archivo a una carpeta.

Este archivo se moverá a la carpeta Documentos, por lo tanto, cambiará de directorio, como podemos observan en la imagen 82.

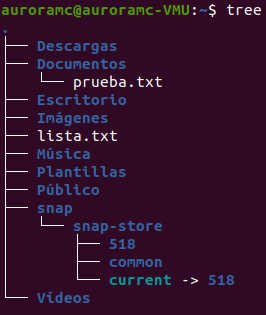


Imagen 82. Directorio actualizado.

* Para mover un directorio es casi lo mismo que mover un archivo, se usa el mismo comando mv. Pero en este caso, tenemos que poner el directorio que queremos mover y el directorio destino.

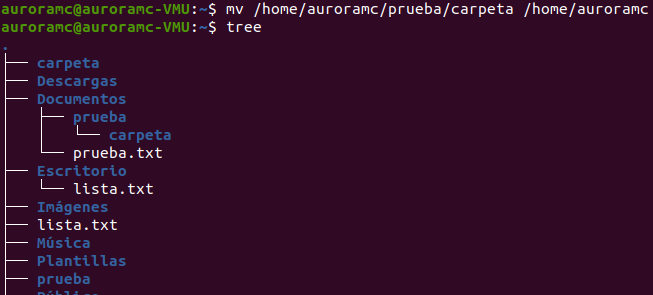


Imagen 83. Captura final del directorio después del comando mv para mover un directorio.

* En este ejemplo vamos a mover varios archivos a otro lugar, para esto se cambia un poco la sintaxis original. Moveremos todos los archivos .txt a la carpeta de Documentos en un solo comando.

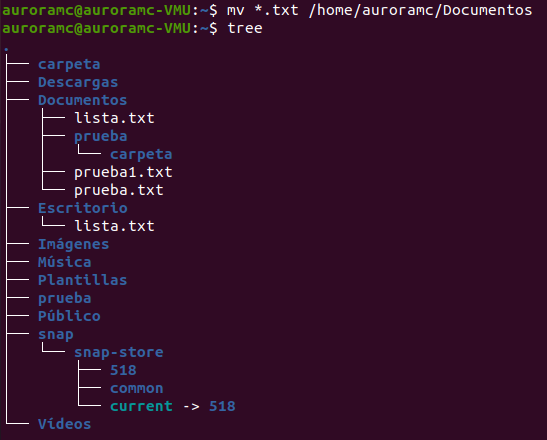


Imagen 84. Captura final del directorio después del comando mv para mover varios archivos.

* Ahora para copiar un archivo vamos a ocupar el comando cp. En este ejemplo copiaremos el archivo lista.txt al escritorio, por lo tanto, este documento estará en dos directorios distintos.

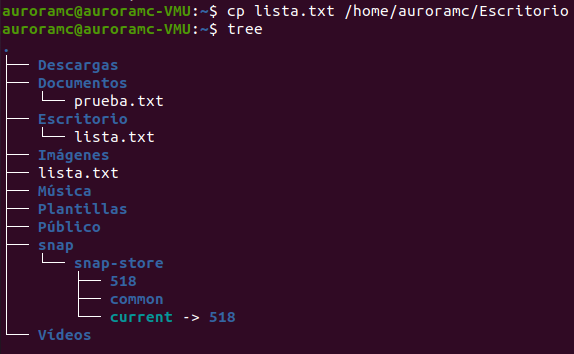


Imagen 85. Uso de cp para copiar un archivo a otro directorio.

* En el siguiente ejemplo vamos a copiar un directorio de un lugar a otro. Es importante estar un nivel abajo del directorio que vamos a copiar, porque si estamos dentro de él es necesario especificar la ruta completa, dado que si solo ponemos el comando de la manera en que la puse, solo va a crear un directorio vacío.

En el comando agregamos -r como atributo para que los elementos internos sean igual copiados.

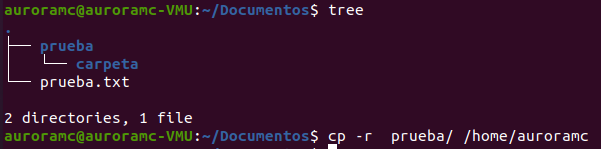


Imagen 86. Directorio de documentos.

De modo que el directorio ya estará copiado en el nuevo.

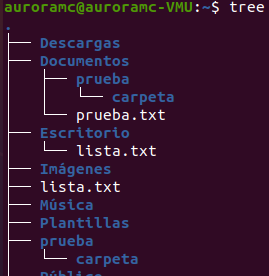


Imagen 87. Captura final del directorio.

6.- Escriba la secuencia para agregar un usuario, verificar que existe, entrar en sesión con el usuario nuevo y después borrarlo cuando el usuario nuevo todavía está en sesión.

* Para crear a un usuario con su directorio home, usamos el siguiente comando:

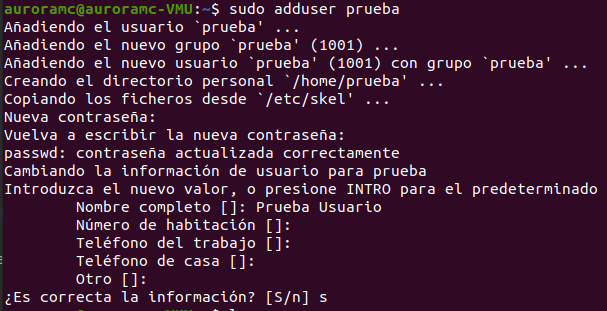


Imagen 88. Comando para crear usuario.

Después de que ingresamos el comando, nos pide información adicional sobre el nuevo usuario que queremos crear y el usuario se llama prueba. Ahora vamos a verificar que si exista el usuario con el siguiente comando. Si nos regresa la información del usuario solicitado es porque si existe, de lo contrario, este no existiría.



Imagen 89. Comando para verificar un usuario.

Luego vamos a iniciar sesión con el usuario nuevo que acabamos de crear.

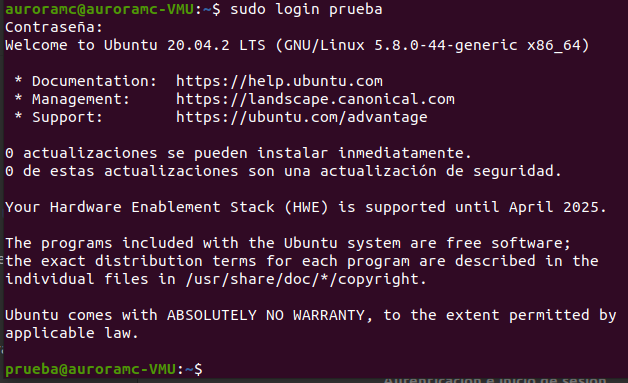


Imagen 90. Inicio de sesión del nuevo usuario.

Por último, vamos a eliminar al usuario que hemos creado con el siguiente comando. Y podemos observar que si queremos verificar si existe, ya no devolverá nada la sentencia.

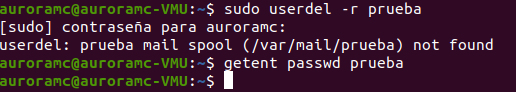


Imagen 91. Eliminación del nuevo usuario.

7.- Escriba la secuencia para crear, compilar y ejecutar "Hola Mundo" en Lenguaje C usando nano.

Primero creamos el archivo, en este caso se llamará ‘HolaMundo.c’, como se muestra en la imagen 92.



Imagen 92. Creación de archivo con nano.

Una vez que demos enter a la sentencia anterior, nos abrirá la ventana del editor de textos, que ofrece nano, en el cual escribiremos el código para imprimir un ‘Hola Mundo’, el cual es mostrado en la imagen 93.

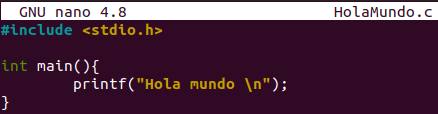


Imagen 93. Código de ‘Hola mundo’ en nano.

Una vez que se haya terminado de escribir el código lo guardamos usando *Ctrl+s* y nos salimos del editor con *Ctrl+x*, ya que nos regresó a la terminal, compilamos el archivo usando *gcc* como lo vemos en la imagen 94.



Imagen 94. Compilación de ‘HolaMundo.c’.

Y por último ejecutamos el programa poniendo *./* antes del nombre de nuestro archivo y ya no ponemos la extensión *.c* y ¡listo! El Hola mundo se visualiza como se muestra en la imagen 95.



Imagen 95. Ejecución de ‘HolaMundo’.

8.- Escriba la secuencia para crear, compilar y ejecutar "Hola Mundo" en Lenguaje C usando vi.

El procedimiento es similar que, con nano, primero se crea el archivo ‘HolaMundo.c’ como se muestra en la imagen 96.



Imagen 96. Creación de archivo con vi.

Una vez que se de enter, nos abrirá la ventana del editor de textos, aquí entra la diferencia con el editor de textos que proporciona nano ya que para escribir en el editor de textos de vi se deben ocupar ciertos comandos, primero se puede poner que se enumeren las líneas usando el comando*: set nu* ya que aparecen podemos comenzar a escribir para lo cual se debe presionar *a* la cual se escribirá de momento es mejor dejarla ahí y escribir la primer línea del código, una vez escrita presionamos *Ctrl+c* para que nos deje volver a navegar por el editor y con las flechas del teclado nos regresamos a donde estaba la ‘a’, nos posicionamos sobre ella y presionamos *x* para que se elimine el carácter, después para generar una nueva línea se debe presionar *o* y volvemos a repetir el procedimiento desde que presionamos *a*, una vez que terminamos de escribir el código como se muestra en la imagen x, ponemos *:wq* para guardar el archivo y cerrar la ventana del editor.

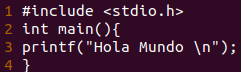


Imagen 97. Código de ‘Hola mundo’ en vi.

Por curiosidad quise volver a abrir el archivo (esto se hace poniendo de nuevo *vi HolaMundo.c*) y apareció el código sin el número de línea que habíamos puesto antes, como se muestra en la imagen 98.

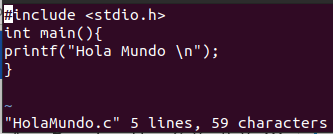


Imagen 98. Visualización del archivo ‘HolaMundo.c’ en vi.

Ahora si una vez que nos salimos de vi compilamos el archivo usando *gcc* como lo vemos en la imagen 99.



Imagen 99. Compilación de ‘HolaMundo.c’.

Y por último ejecutamos el programa poniendo *./* antes del nombre de nuestro archivo y ya no ponemos la extensión *.c* y terminamos visualizando Hola Mundo como se muestra en la imagen 100.



Imagen 100. Ejecución de ‘HolaMundo’.

9.- Escriba la secuencia para crear, compilar y ejecutar "Hola Mundo" en Lenguaje C usando pico.

Pico es un sencillo editor de texto basado en el editor del sistema de mensajes Pine. Al igual que en Pine, los comandos son desplegados en la parte inferior de la pantalla, y se proporciona ayuda sensible al contexto. Conforme los caracteres son tecleados se insertan inmediatamente en el texto. Sin embargo, este editor de texto ya no está disponible, ya que se volvió la base para nano y esto se nota al poner la sentencia *pico HolaMundo.c*, dicho que abre la ventana del editor de texto de nano, por lo que sería el mismo procedimiento que se usó con nano.

Se hizo el intento de instalar pico, pero devolvió un mensaje de que dicho paquete no está disponible como se muestra en la imagen 101, además que menciona que el paquete de pico lo reemplaza el paquete nano.

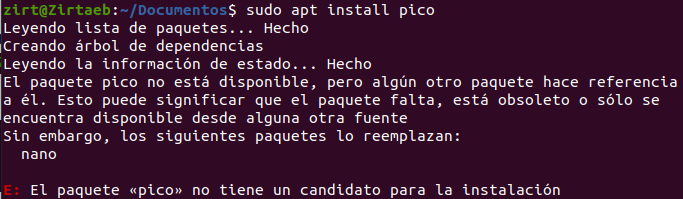


Imagen 101. Intentó de instalación de pico

10.- Usar la función getenv( ) en un programa en lenguaje C (Programa1a.c) para que imprima únicamente el valor de la variable HOME.

La función getenv() busca la variable de entorno apuntada por el nombre y devuelve el valor asociado a la variable y si no existe la variable de entorno, devuelve NULL.

En este caso solo se imprimirá el valor de la variable HOME y el código para hacerlo es el que se muestra en la imagen 102.

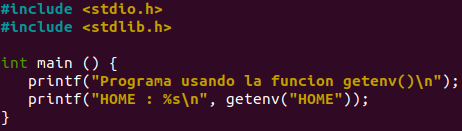


Imagen 102. Código de uso de la función getenv().

Una vez que compilamos y ejecutamos el archivo nos mostrara que home tiene el valor de */home/zirt* como se muestra en la imagen 103.

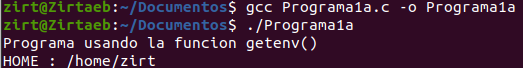


Imagen 103. Compilación y ejecución del archivo ‘Programa1a.c’.

11.- Realizar un programa en lenguaje C que realice las siguientes operaciones (llame al programa Programa2a.c):

• Crear un archivo

• Introducir diez números dados por el usuario

• Grabar los números del punto anterior en el archivo creado

• Cerrar en archivo

El programa constó sólo de la función principal en adición con la función encargada de escribir el archivo.

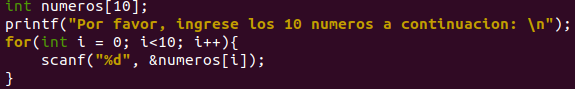
Lo primero que nos encontramos es la lectura de los 10 números que está en la función principal (main), y se almacenan en un arreglo de enteros y es lo que hacen las líneas mostradas en la siguiente imagen.

Imagen 104. Código donde se piden al usuario los 10 números.

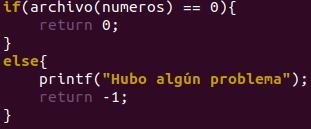
Más tarde ejecutamos la función llamada *archivo,* le pasamos como argumento el arreglo de números, lo encerramos en un if-else para comprobar que no haya errores.

Imagen 105. Código donde se llama a la función archivo.

Ahora se ejecuta la función archivo, primero se crea un puntero al archivo de modo que cree el archivo para escribir sobre él, más tarde iteramos el arreglo de números, los convertirnos a una cadena y agregamos un salto de línea para que distingan muy bien en el archivo de texto. Al final de cada iteración vamos agregar esa cadena al archivo que estamos apuntando.

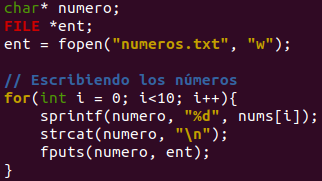
Ahora ilustro lo anterior.

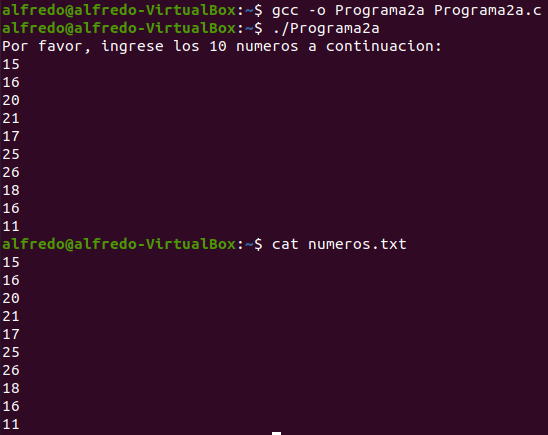
Imagen 106. Código donde abrimos y escribimos los 10 números.

Por últimos sólo cerramos el archivo y devolvemos el valor de la comprobación de si hay o no algún error. En la siguiente imagen se aprecia.

Imagen 107. Código donde cerramos el archivo.

Y con eso hemos cumplido todos los puntos que se han planteado en el problema, por último, vamos hacer la prueba de que todo funcione correctamente con los números: 15, 16, 20, 21, 17, 25, 26, 18, 16, 11.

A continuación, adjuntamos las imágenes de la prueba realizada a la aplicación.

Imagen 108. Compilación, ejecución, inserción de números y visualización del archivo.

Para encontrar el código completo ir a la sección de anexos en el punto 1.1.

12.- Realizar un programa que realice las siguientes operaciones (llame al programa Programa3a.c):

• Abrir el archivo del programa anterior (Programa2a.c) en modo lectura

• Leer los números del archivo

• Mostrar los números en pantalla

• Calcular promedio, suma, multiplicación, desviación estándar y varianza de los números

• Mostrar los resultados en pantalla

• Cerrar el archivo

En resumen, primero hicimos la apertura del archivo y la lectura de una manera bastante sencilla, son funciones bien conocidas cuando se trata de manejar archivos, y dentro de la lectura, se leyeron cada línea, recordemos que los números fueron guardados uno por línea, y al mismo tiempo convertimos la cadena con los números a números enteros y los guardamos en un arreglo para su posterior manejo con las operaciones pedidas y cerramos el archivo puesto que ya tenemos los números.

Ahora explicaré el procedimiento de ejecución:

Si ejecutamos el Programa2a.c y guardamos los siguientes datos:

52,74,9,123,7,14,63,99,74,233

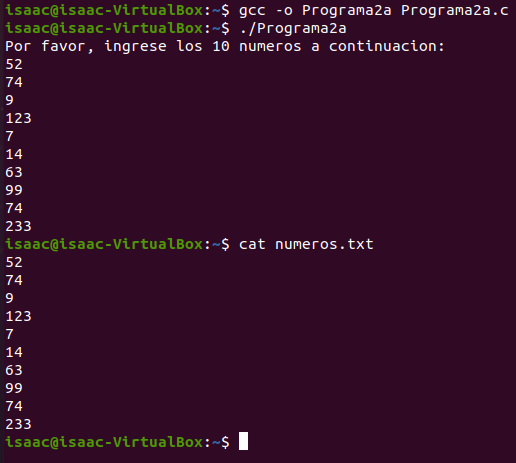


Imagen 109. Ejecución del Programa2a.c

Si abrimos el archivo vemos como si se guardaron los datos:

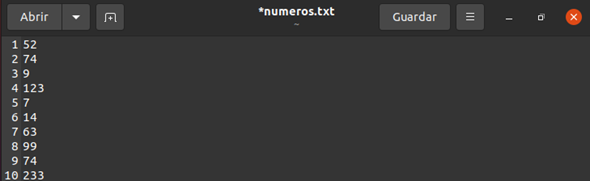


Imagen 110.El archivo “numeros.txt.

Ahora si ejecutamos el Programa3a.c obtenemos lo siguiente:

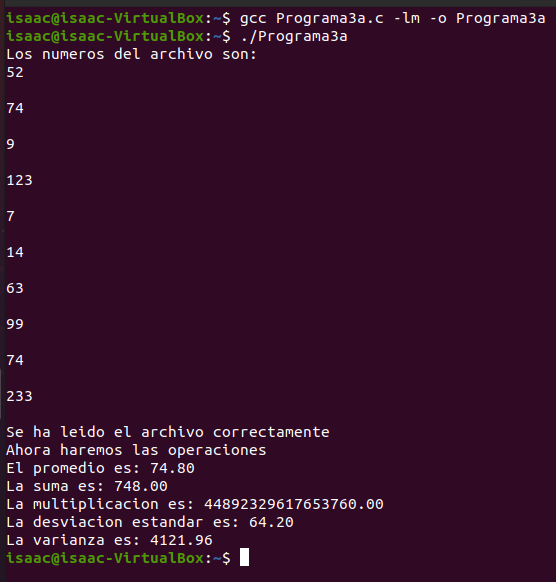


Imagen 111. Ejecución del Programa3a.c

Para demostrar el resultado de la desviación estándar, vemos que el siguiente solver muestra el mismo resultado que el programa:



Imagen 112. Solver para desviación estandar.

Ahora explicaré un poco más a detalle el código:

Primero se crea un puntero la variable tipo FILE que se llama flujo y utilizamos la función fopen para abrir el archivo, en este caso “números.txt” en modo lectura (“r”). Utilizamos un condicional para que en caso de que el archivo esté vacío nos muestre “Error en la apertura del archivo”

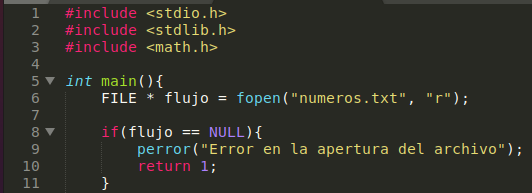


Imagen 113. Programa3a.c primera parte.

Lo siguiente es crear un arreglo de caracteres de 10 posiciones y un arreglo de enteros igual de 10 posiciones, estos los usaremos más adelantes.

Después dentro de un while, utilizamos la función fgets que se encarga de leer y almacenar una cadena de caracteres, la variable es guarda hasta que haya un saldo de línea hasta que haya un saldo de línea, el primer parámetro nos dice en donde se quiere copiar la línea leída (en el arreglo de caracteres), luego en máximo número de caracteres, yo le puse 10, el tercer parámetro es para indicar que se trata de una variable FILE.  
 Imprimimos cada línea (numero). La funcon atoi se encarga de hacer el casteo de carácter a entero, entonces a la primera posición del arreglo de enteros se le asigna la primera posición del arreglo de caracteres, y así con todas las posiciones.

Cerramos el archivo e indicamos que se ha leído correctamente el archivo.

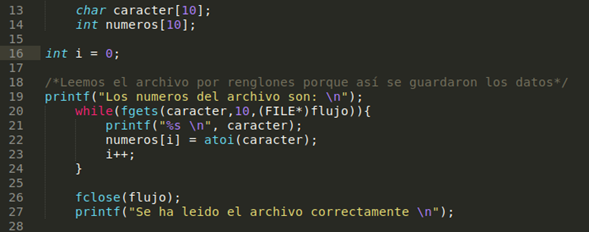


Imagen 114. Programa3a.c segunda parte.

Lo anterior era lo más complicado de hacer en el programa, lo demás solo es hacer las operaciones con ayuda de bucles.

Algo interesante que quiero comentar es que al utilizar funciones de la librería math.h me salía el siguiente error:

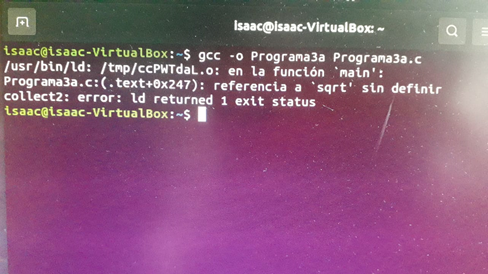


Imagen 115. Error en Linux con la librería math.h

Y al investigar encontré que algunos compiladores nativos del sistema de Linux no se enlazan con la librería math.h si no le indicamos explícitamente, entonces al gcc hay que agregarle -lm que significa librería math para solucionar el problema.



Imagen 116. Arreglo del problema con math.h

# **Códigos y ventanas de ejecución**

## Punto 11 del Desarrollo

Código completo

#include <stdio.h>

#include <string.h>

int archivo(int[]);

// Aquí se ingresan los números y se mandan a escribir

int main(void){

int numeros[10];

printf("Por favor, ingrese los 10 numeros a continuacion: \n");

for(int i = 0; i<10; i++){

scanf("%d", &numeros[i]);

}

if(archivo(numeros) == 0){

return 0;

}

else{

printf("Hubo algún problema");

return -1;

}

}

int archivo(int nums[]){ // Lectura de archivo y escritura de numeros

char\* numero;

FILE \*ent;

ent = fopen("numeros.txt", "w");

// Escribiendo los números

for(int i = 0; i<10; i++){

sprintf(numero, "%d", nums[i]);

strcat(numero, "\n");

fputs(numero, ent);

}

int error = ferror(ent);

fclose(ent);

return error;

}

## Punto 12 del Desarrollo

Código completo

#include <stdio.h>

#include <stdlib.h>

#include <math.h>

int main(){

FILE \* flujo = fopen("numeros.txt", "r");

if(flujo == NULL){

perror("Error en la apertura del archivo");

return 1;

}

char caracter[10];

int numeros[10];

int i = 0;

/\*Leemos el archivo por renglones porque así se guardaron los datos\*/

printf("Los numeros del archivo son: \n");

while(fgets(caracter,10,(FILE\*)flujo)){

printf("%s \n", caracter);

numeros[i] = atoi(caracter);

i++;

}

fclose(flujo);

printf("Se ha leido el archivo correctamente \n");

int j = 0;

printf("Ahora haremos las operaciones \n");

/\*Hacemos la suma de los numeros\*/

float suma;

suma = numeros[0];

j = 1;

while(j<10){

suma += numeros[j];

j++;

}

/\*Hacemos el promedio de los numeros\*/

float promedio = suma/10;

printf("El promedio es: %.2f\n", promedio);

printf("La suma es: %.2f\n", suma);

/\*Hacemos la multiplifacion de los numeros\*/

float multiplicacion;

multiplicacion = numeros[0];

j = 1;

while(j<10){

multiplicacion \*= numeros[j];

j++;

}

printf("La multiplicacion es: %.2f\n", multiplicacion);

/\*Hacemos la varianza\*/

/\*Nota: estoy calculando la varianza para una POBLACION, no para una muestra\*/

/\*Primero hacemos la sumatoria\*/

float sumatoria;

float ope;

float ope2;

j = 0;

while(j<10){

ope = numeros[j] - promedio;

ope2 = ope\*ope;

sumatoria += ope2;

ope2 = 0;

j++;

}

/\*Dividimos la sumatoria entre el numero de objetos\*/

float varianza = sumatoria/10;

float desviacion = sqrt(varianza);

//printf("La sumatoria es %.2f\n",sumatoria );

printf("La desviacion estandar es: %.2f\n", desviacion);

printf("La varianza es: %.2f\n", varianza);

}

# **Bibliografía**

1. Noguera, B.(S/F). ¿Qué es Genome? [En línea]. Available: <https://culturacion.com/que-es-gnome/>
2. Noguera, B.(S/F). ¿Qué es KDE? [En línea]. Available: <https://culturacion.com/que-es-kde/>
3. Méndez, A. (2014). Genome vs KDE. [En línea]. Available: <https://alberto.mendezcabrera.com/gnome-vs-kde-desde-el-punto-de-vista-del-usuario/>
4. Oriol (2016). Como Gestionar Usuarios, Grupos y Permisos en Linux. [En línea]. Available: <https://computernewage.com/2016/05/22/gestionar-usuarios-y-permisos-en-linux/>.
5. S/A, (2021). ¿Qué es una GUI? [En línea]. Available: <https://www.ionos.mx/digitalguide/paginas-web/desarrollo-web/que-es-una-gui/>
6. ZEOKAT (2018). Ruta relativa y absoluta, diferencias y características. [Online]. Available: <https://www.vozidea.com/diferencias-ruta-relativa-y-absoluta>
7. S/A (2019). Rutas relativas y rutas absolutas. De novato a novato. [Online]. Available: <https://denovatoanovato.net/rutas-relativas-y-rutas-absolutas/>
8. Yeraldine (2017). Redirecciones y tuberías en Linux. Profesional Review [Online]. Available: <https://www.profesionalreview.com/2017/02/19/redirecciones-tuberias-linux/#:~:text=Las%20redirecciones%20consisten%20en%20trasladar,eso%20haciendo%20uso%20del%20s%C3%ADmbolo%20%3E.>
9. Nacx (S/F). Redirecciones y pipes. Adslayuda. [Online]. Available: <https://www.adslayuda.com/linux-redirecciones.html>
10. 1&1 IONOS Inc. (2020) Los comandos de Linux más importantes. [En línea] Recuperado de: <https://www.ionos.mx/digitalguide/servidores/configuracion/comandos-de-linux-la-lista-fundamental/>
11. Deyimar, A. (2020) Variables de entorno de Linux: cómo leerlas y configurarlas en un VPS de Linux. [En línea] Recuperado de: <https://www.hostinger.es/tutoriales/variables-de-entorno-linux-como-leerlas-y-configurarlas-vps>
12. R. Velasco. (2021, Enero 12). Aprende a dominar la Terminal de Linux como un profesional [En linea]. Available:<https://www.softzone.es/programas/linux/terminal-linux/>
13. R. Morales. (2013, Mayo 8). Tipos de usuarios y grupos, directorios e IDs. [En linea]. Available: <https://www.ticarte.com/contenido/tipos-de-usuarios-y-grupos-directorios-e-ids>
14. 1&1 IONOS Inc. (2020) Los comandos de Linux más importantes. [En línea] Recuperado de: <https://www.ionos.mx/digitalguide/servidores/configuracion/comandos-de-linux-la-lista-fundamental/>
15. Deyimar, A. (2020) Variables de entorno de Linux: cómo leerlas y configurarlas en un VPS de Linux. [En línea] Recuperado de: <https://www.hostinger.es/tutoriales/variables-de-entorno-linux-como-leerlas-y-configurarlas-vps>
16. R. Velasco. (2021, Enero 12). Aprende a dominar la Terminal de Linux como un profesional [En linea]. Available: <https://www.softzone.es/programas/linux/terminal-linux/>
17. R. Morales. (2013, Mayo 8). Tipos de usuarios y grupos, directorios e IDs. [En linea]. Available: <https://www.ticarte.com/contenido/tipos-de-usuarios-y-grupos-directorios-e-ids>
18. Anonimo. (2015). ¿Cuál es la diferencia entre Bash, Zsh y otros shells de Linux?. [En linea]. Available:<https://es.phhsnews.com/what-s-difference-between-bash-zsh-and-other-linux-shells3733>
19. Computer Hope (2020). Linux Vi Command – Computer Hope. [Online]. Available: <https://www.computerhope.com/unix/uvi.htm>
20. Computer Hope (2019). Linux less command– Computer Hope. [Online]. Available: <https://www.computerhope.com/unix/uless.htm>
21. Computer Hope (2019). Linux more Command – Computer Hope. [Online]. Available: <https://www.computerhope.com/unix/umore.htm>
22. S. Javier (2006). Tutorial Básico de GNU/Linux: Caracteres “comodín”. [Online]. Available: [https://smaldone.com.ar/documentos/misdocs/tutorial-gnu-linux/index-6.html#:~:text=El%20comod%C3%ADn%20%E2%80%9C%20\*%20%E2%80%9D%20hace%20referencia,al%20cual%20nos%20estamos%20refiriendo.](https://smaldone.com.ar/documentos/misdocs/tutorial-gnu-linux/index-6.html#:~:text=El%20comod%C3%ADn%20%E2%80%9C%20*%20%E2%80%9D%20hace%20referencia,al%20cual%20nos%20estamos%20refiriendo.)
23. Oracle corporation (2010). Resumen de los comandos básicos de vi. [En linea]. Available: <https://docs.oracle.com/cd/E19620-01/805-7644/6j76klopr/index.html>
24. V. C. Virginia (S/F). Manejo de archivos en C. [Online]. Available: <https://w3.ual.es/~abecerra/ID/archivos.pdf>