UNIVERSIDAD NACIONAL DE LA MATANZA

Departamento de Ingeniería e Investigaciones Tecnológicas

Sistemas Operativos (Plan 2009)

Jefe de Cátedra: Fabio E. Rivalta
Equipo de Docentes: Boettner F., Catalano L., de Lizarralde R, Villamayor A.
Auxiliares docentes: Loiacono F., Hirschfeldt D., Piubel F., Rodriguez A., Segura L., Fernandez Piñeiro, Radice A.

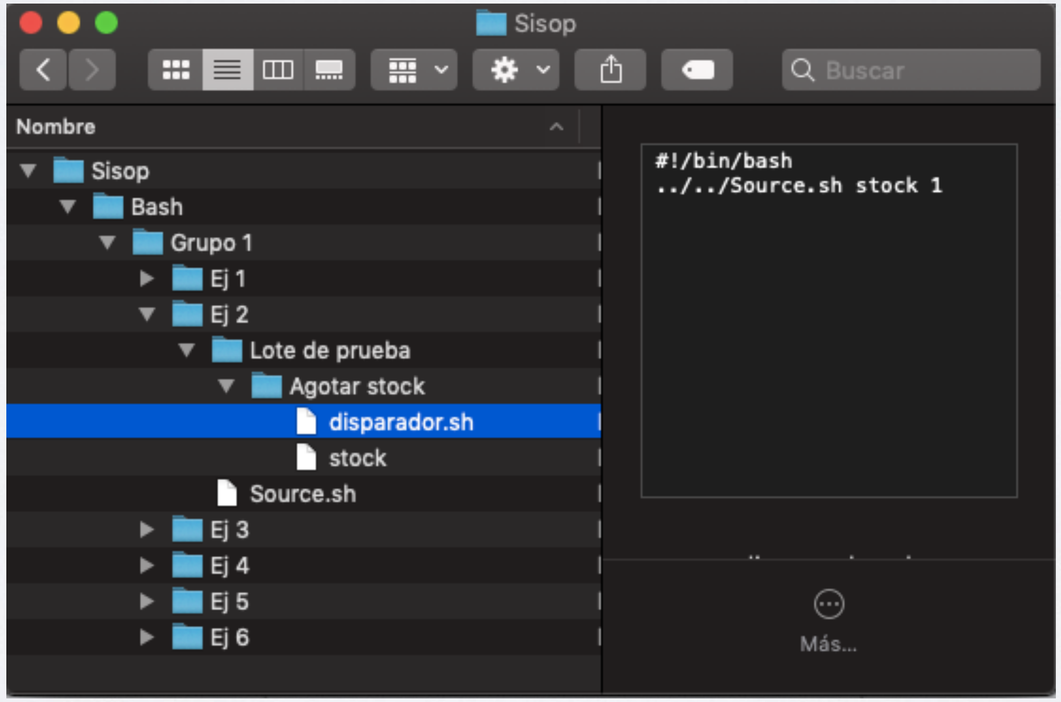
SCRIPTING -> BASH

SISTEMAS OPERATIVOS UNLAM.

Autor: Radice Adrian

¿CÓMO ENTREGAR EL PRÁCTICO?

- Carátula impresa por TP. (Completar sus campos).
- Archivo comprimido con el siguiente contenido:
  - Una carpeta por ejercicio, su titulo será Ej X.
  - Es obligatorio que cada ejercicio incluya un lote de prueba propio.
  - Él mismo se compondrá de la entrada y un script disparador para la ejecución del mismo.



Ejemplo

ADVERTENCIA

TODO EJERCICIO DEBE FUNCIONAR EN EL LABORATORIO

ENTORNO DE LABORATORIO

LUbuntu 18.0.4

Autor: Radice Adrian

PREPARANDO NUESTRO ENTORNO DE TRABAJO

ALTERNATIVAS

Instalar Nativamente Alguna Distribución de GNU/LINUX

Recuerda que si vas instalar alguna distribución en un equipo en que ya tenemos instalado otro sistema operativo, y queremos conservar ambos vamos a requerir de un gestor de arranque, por ejemplo GRUB

GESTOR DE ARRANQUE GRUB

RECUERDA QUE NO ES NECESARIO TENER MÁS DE UN DISCO, PUEDE PARTICIONARSE UN DISCO Y TENER DOS O MAS SISTEMAS EN EL.

OTRA ALTERNATIVA

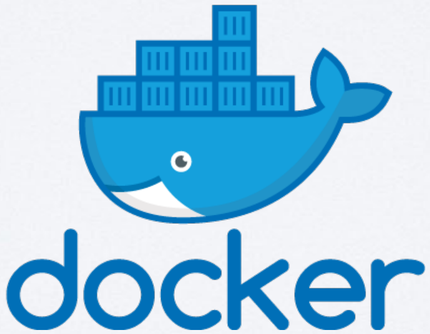
UTILIZAR UNA MAQUINA VIRTUAL

Recomendada por la cátedra

Autor: Radice Adrian

¿QUÉ HERRAMIENTA NECESITAMOS PARA UTILIZAR UNA VM?

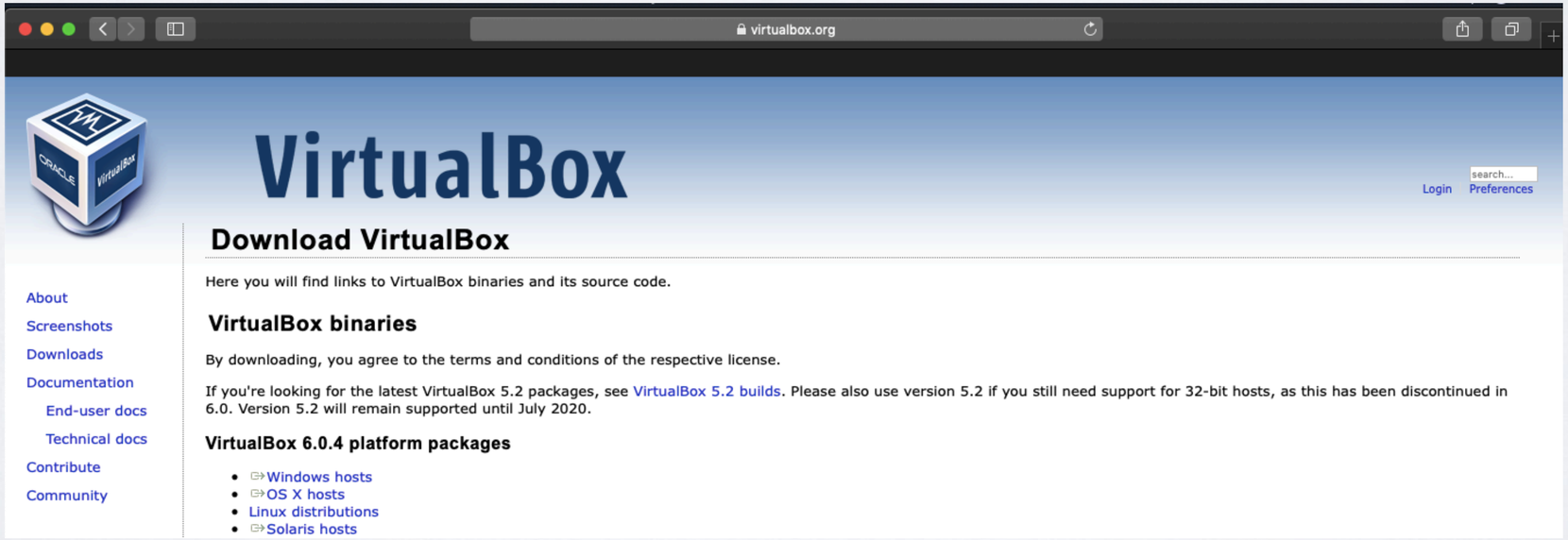
NECESITAMOS ALGÚN ADMINISTRADOR DE VM



PREPAREMOS UN ENTORNO EN UNA VM UTILIZANDO VIRTUAL BOX

DESCARGAMOS VIRTUAL BOX PARA NUESTRO SISTEMA OPERATIVO

https://www.virtualbox.org/wiki/Downloads

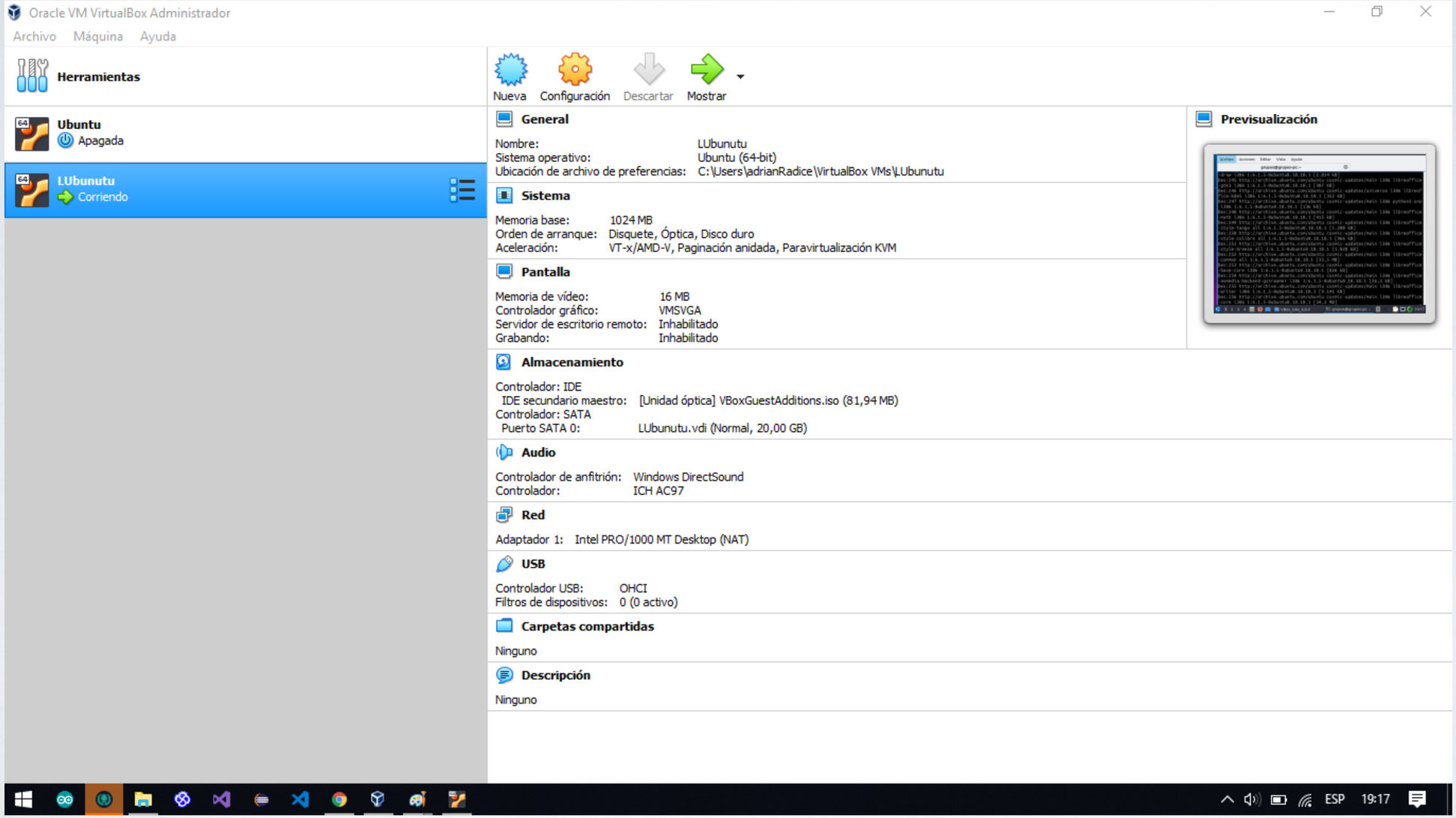


Autor: Radice Adrian

INSTALAMOS

Siguiente, Siguiente, ….. Siguiente, Finalizar.

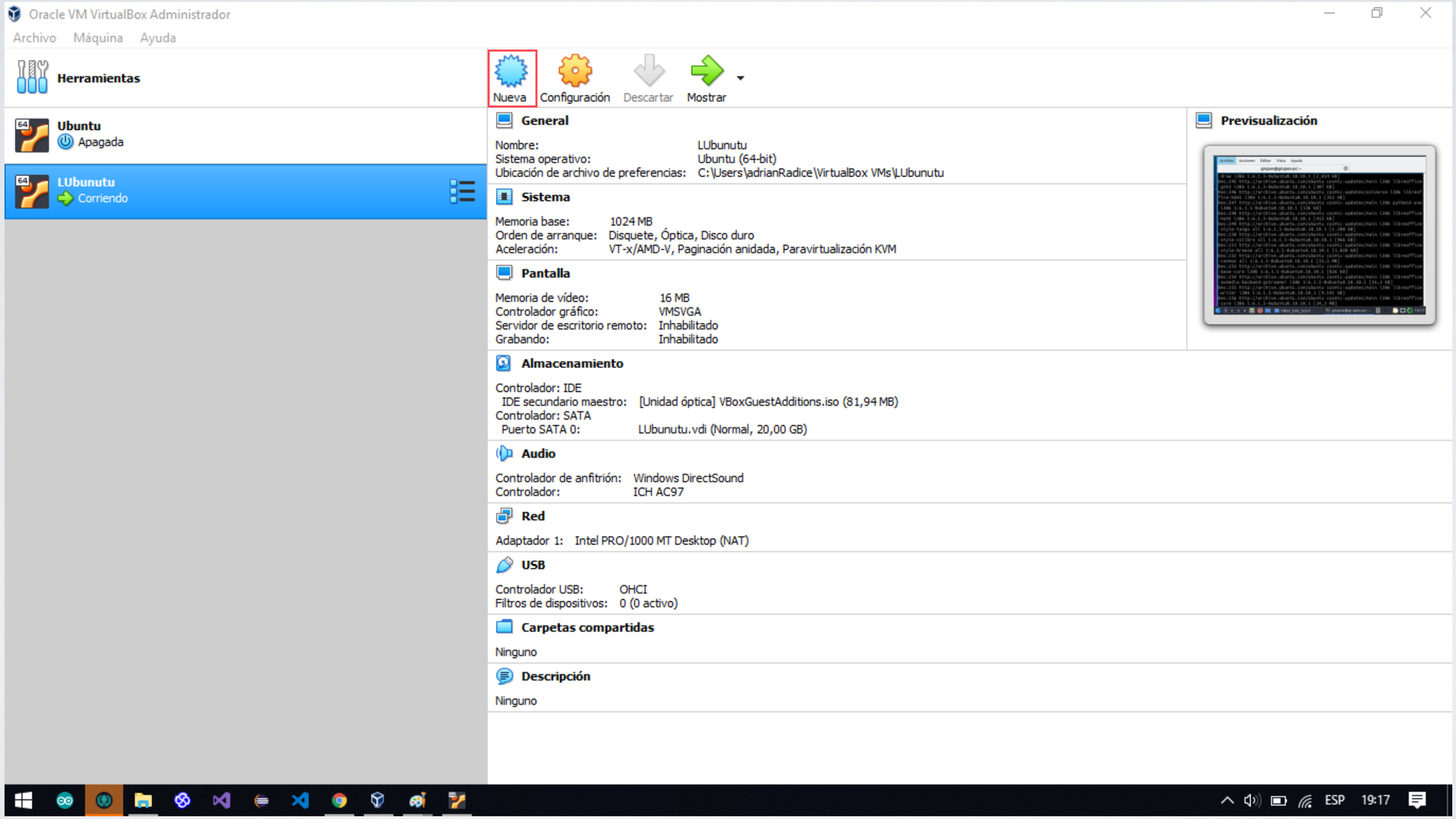
ABRIMOS VIRTUAL BOX



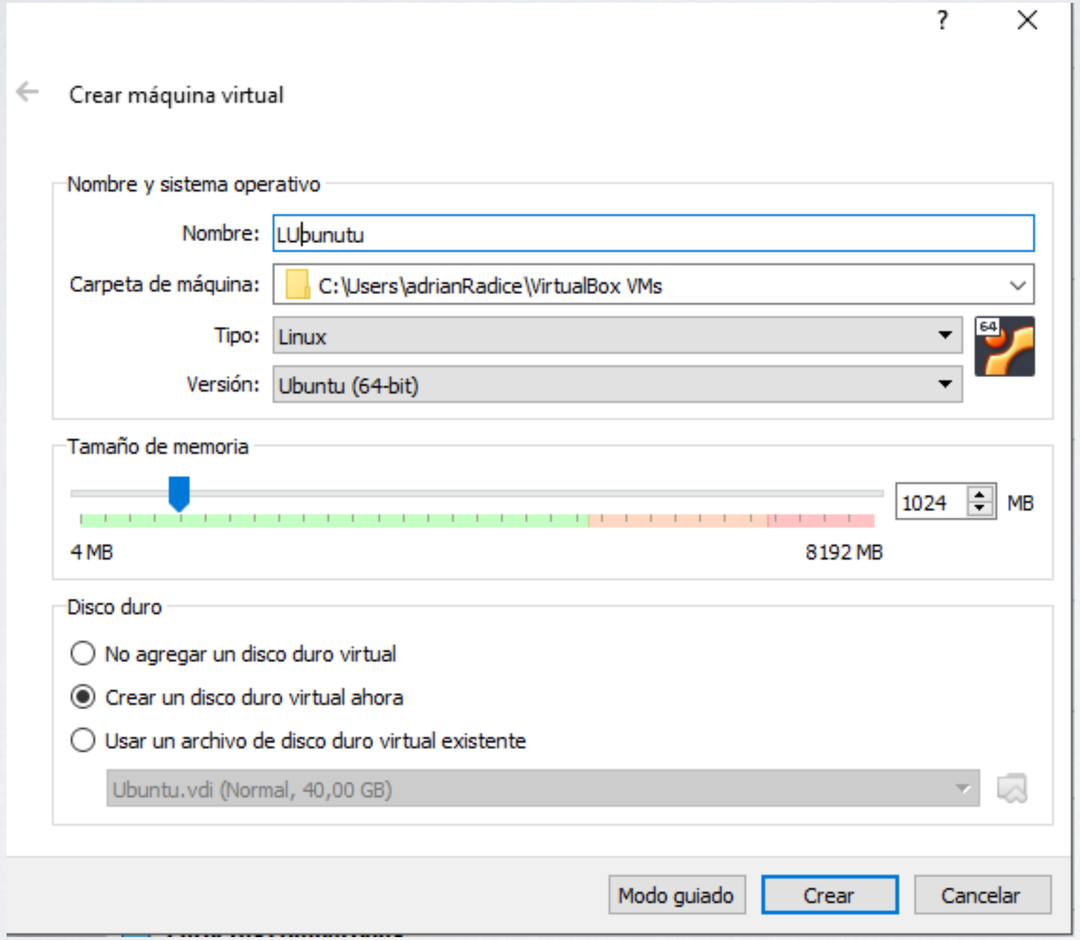
Autor: Radice Adrian

AHORA CREAMOS UNA VM

SELECCIONAMOS NUEVA

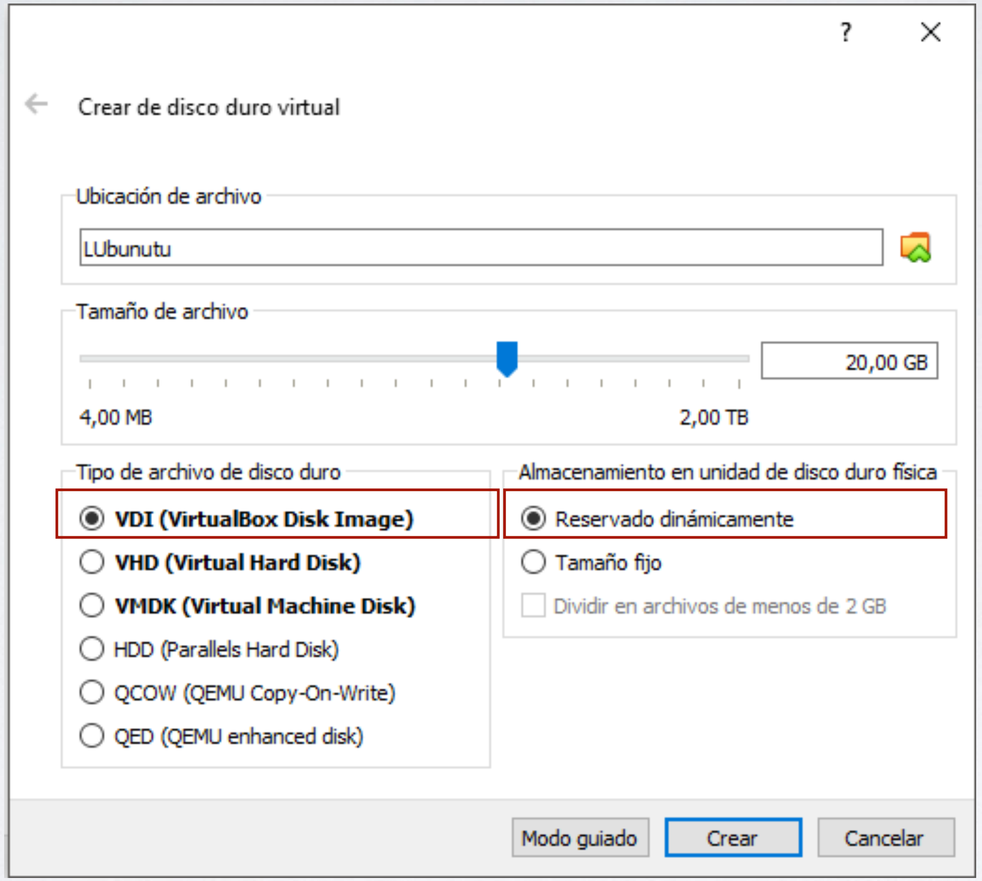


COMPLETAMOS LOS CAMPOS



Mínimo 512 MB

COMPLETAMOS LOS CAMPOS



Mínimo 2GB

HASTA AHORA SOLO TENEMOS UNA VM.

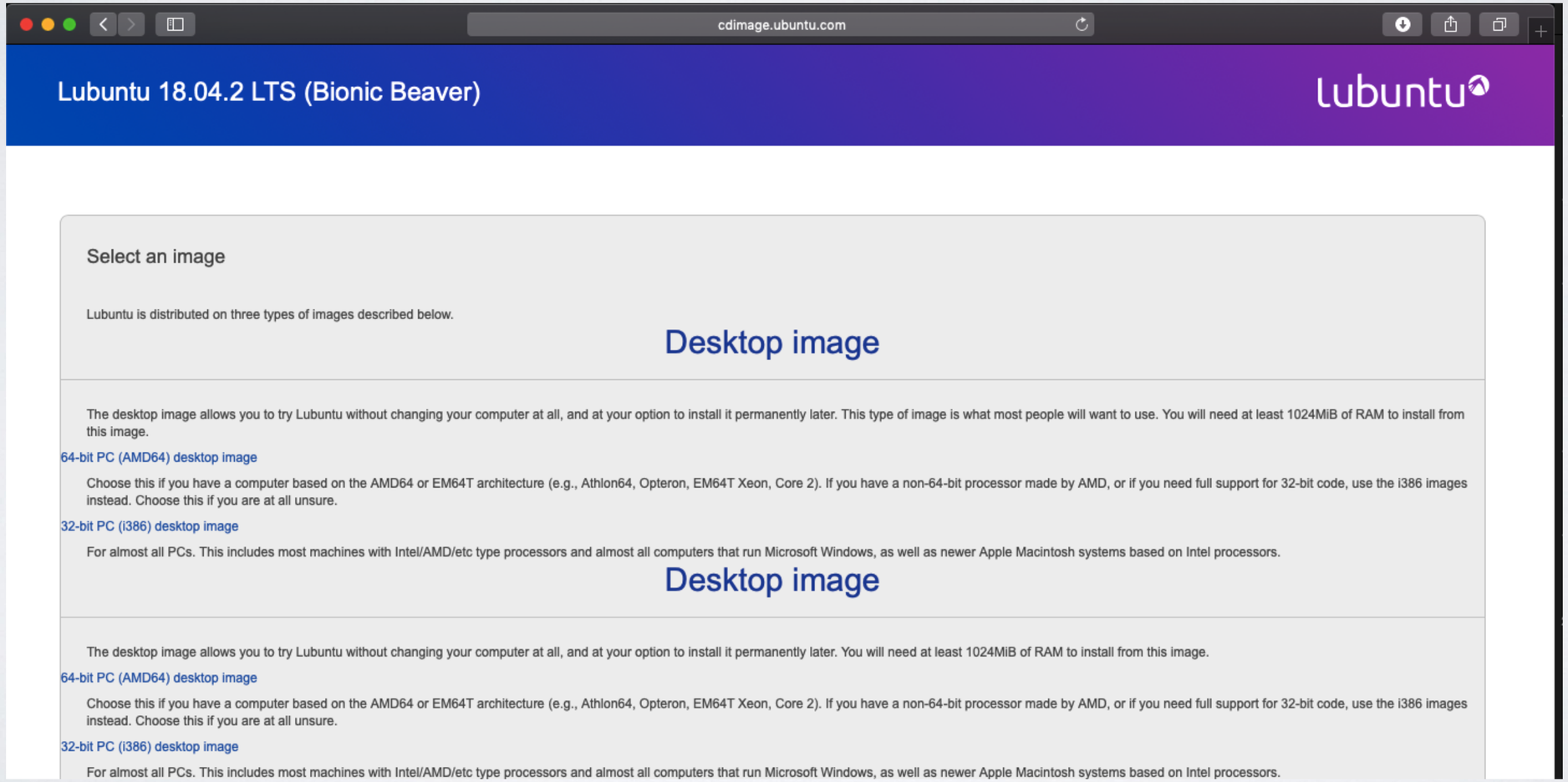
Falta indicar el sistema operativo que ejecutara

INSTALANDO LUBUNTU

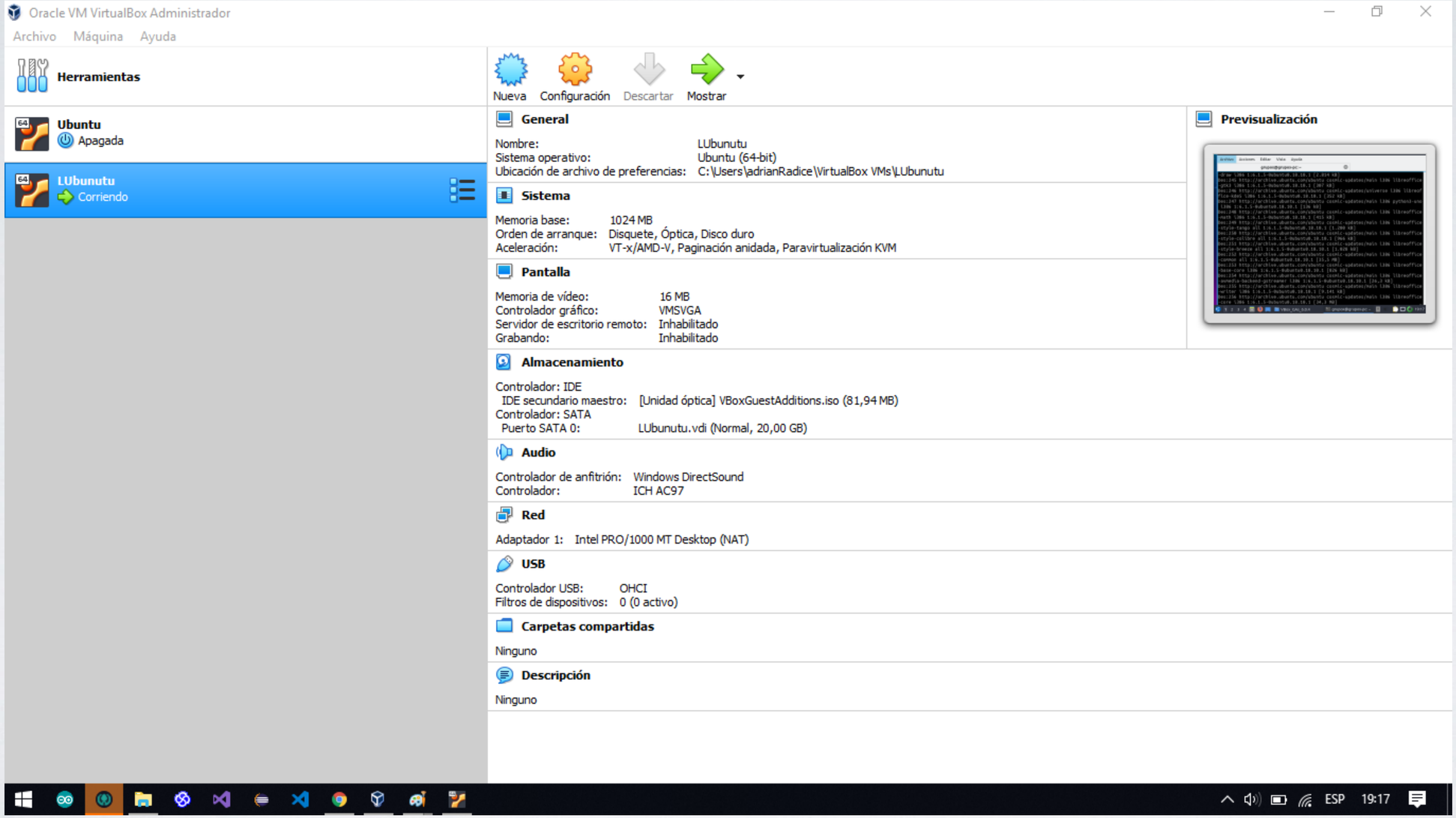
Autor: Radice Adrian

PRIMERO DESCARGAMOS LA ISO

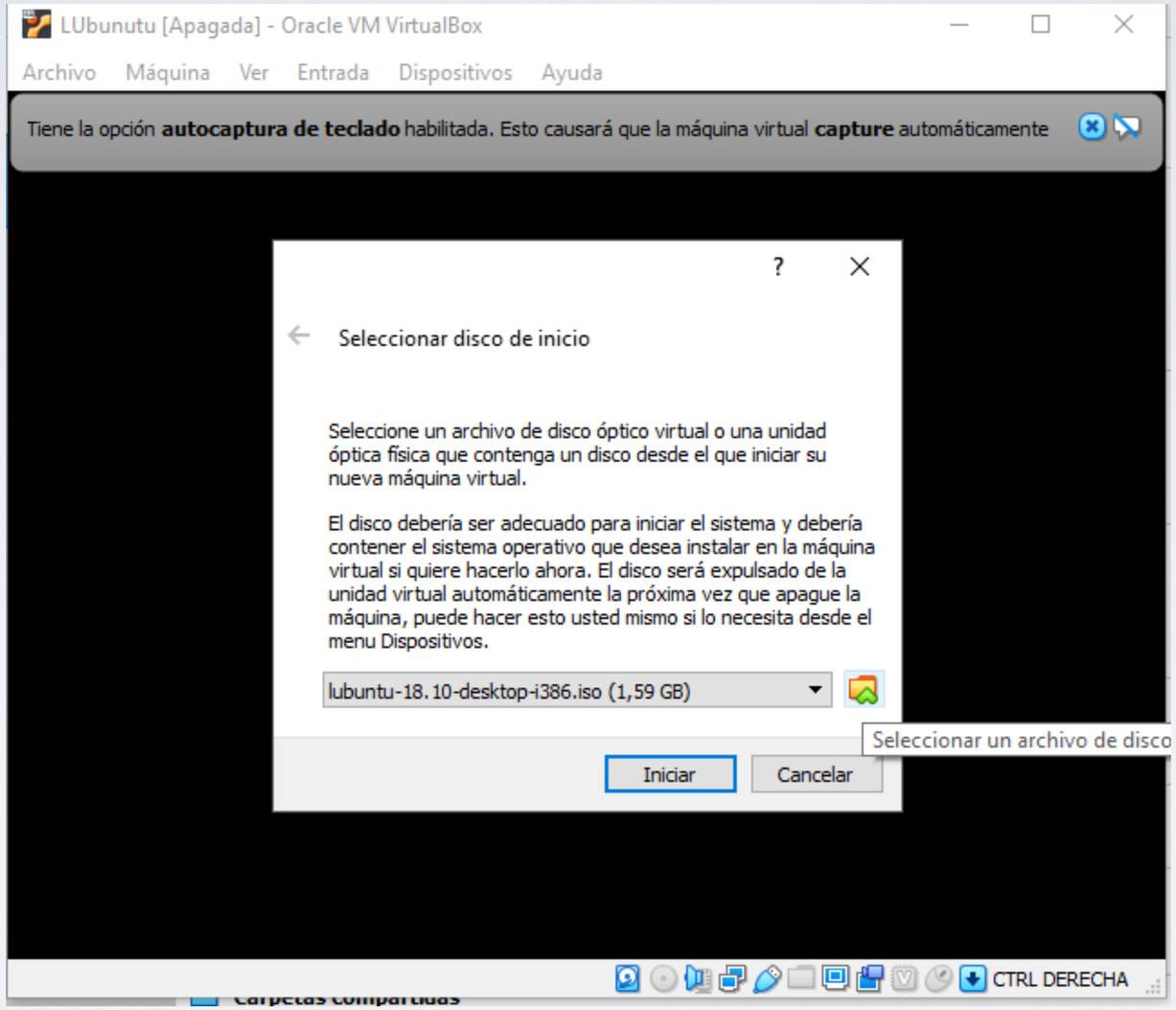
http://cdimage.ubuntu.com/lubuntu/releases/18.04/release/



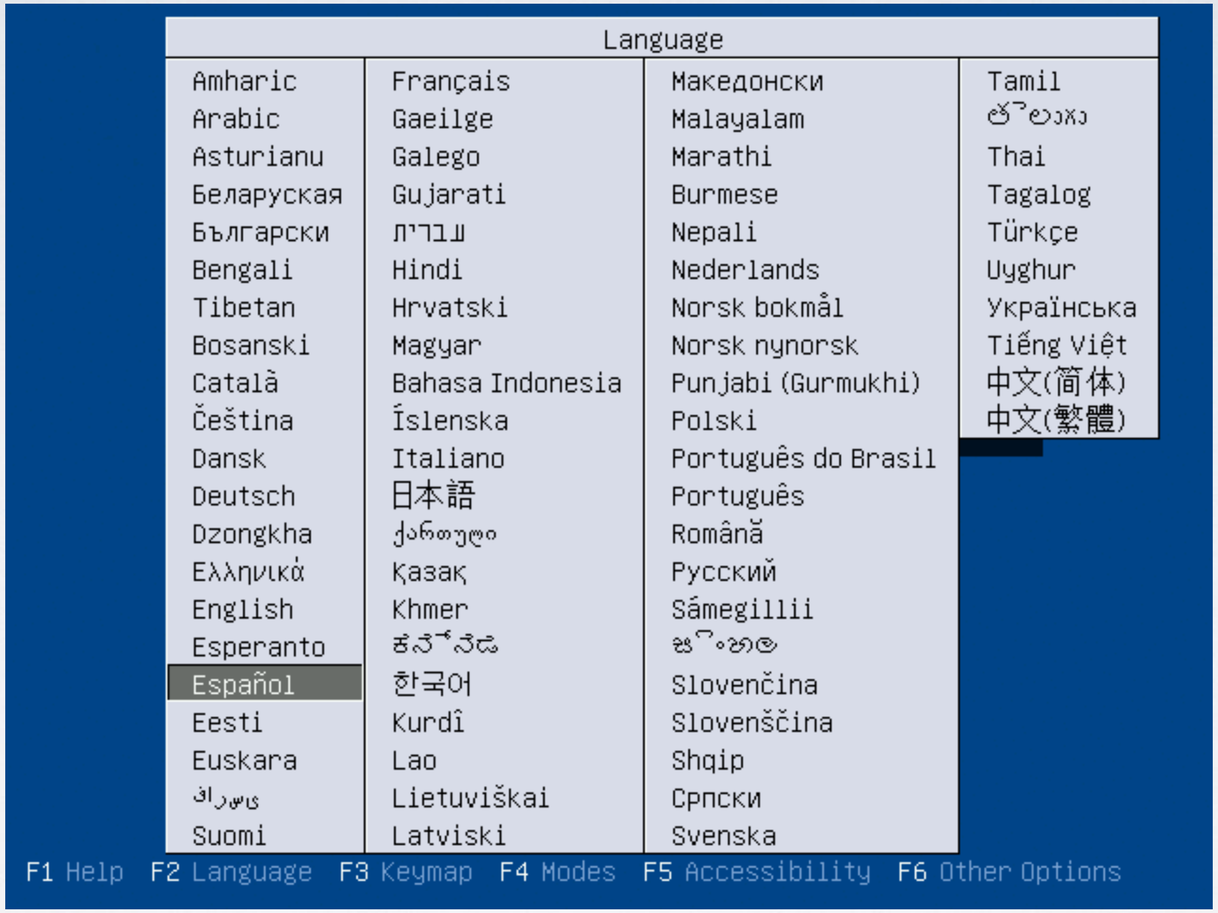
Ahora debemos arrancar nuestra VM, haciendo doble click sobre ella



Seleccionamos la ISO descargada y presionamos iniciar

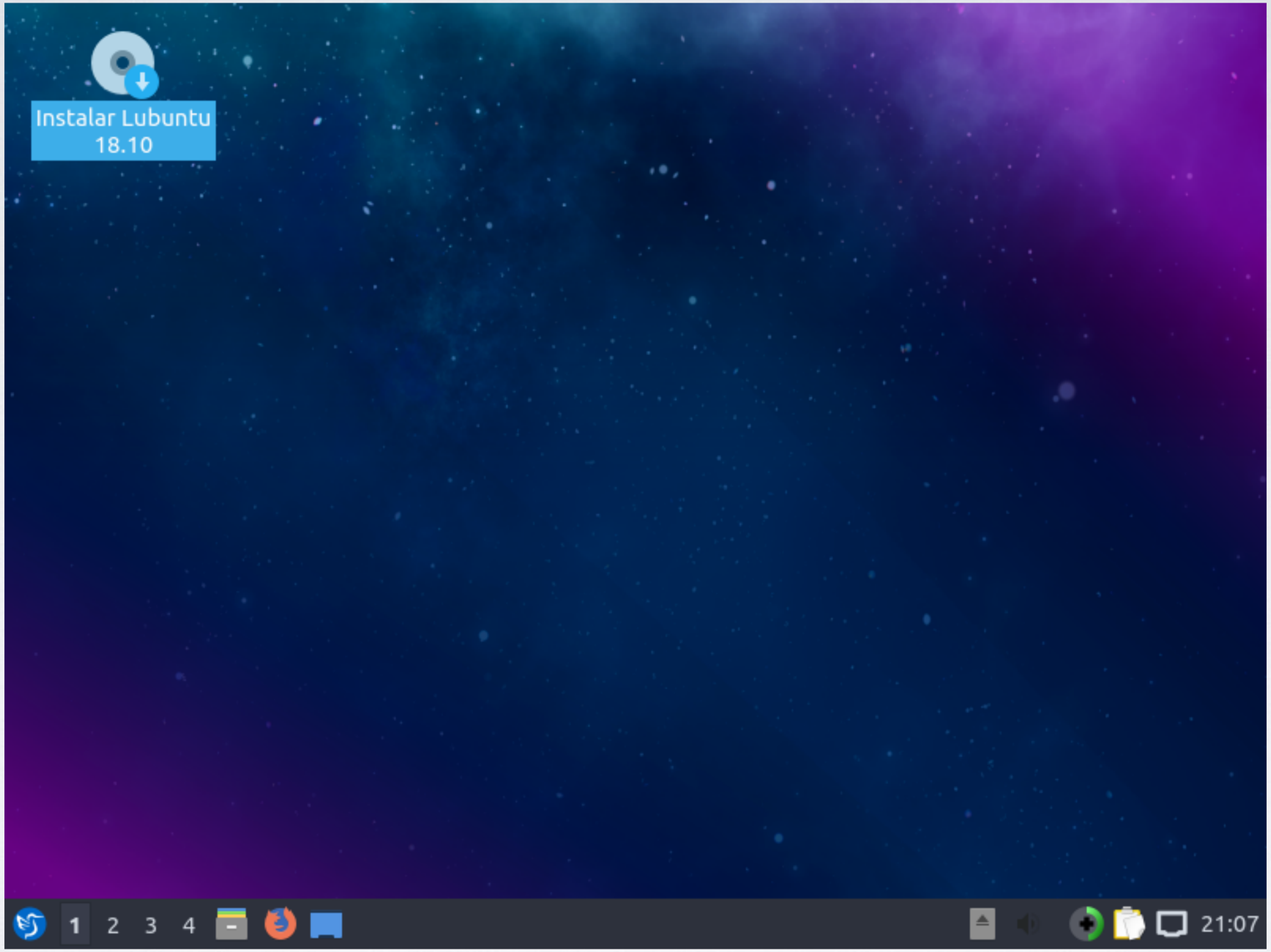


AHORA DEBEMOS INSTALAR EL SISTEMA OPERATIVO

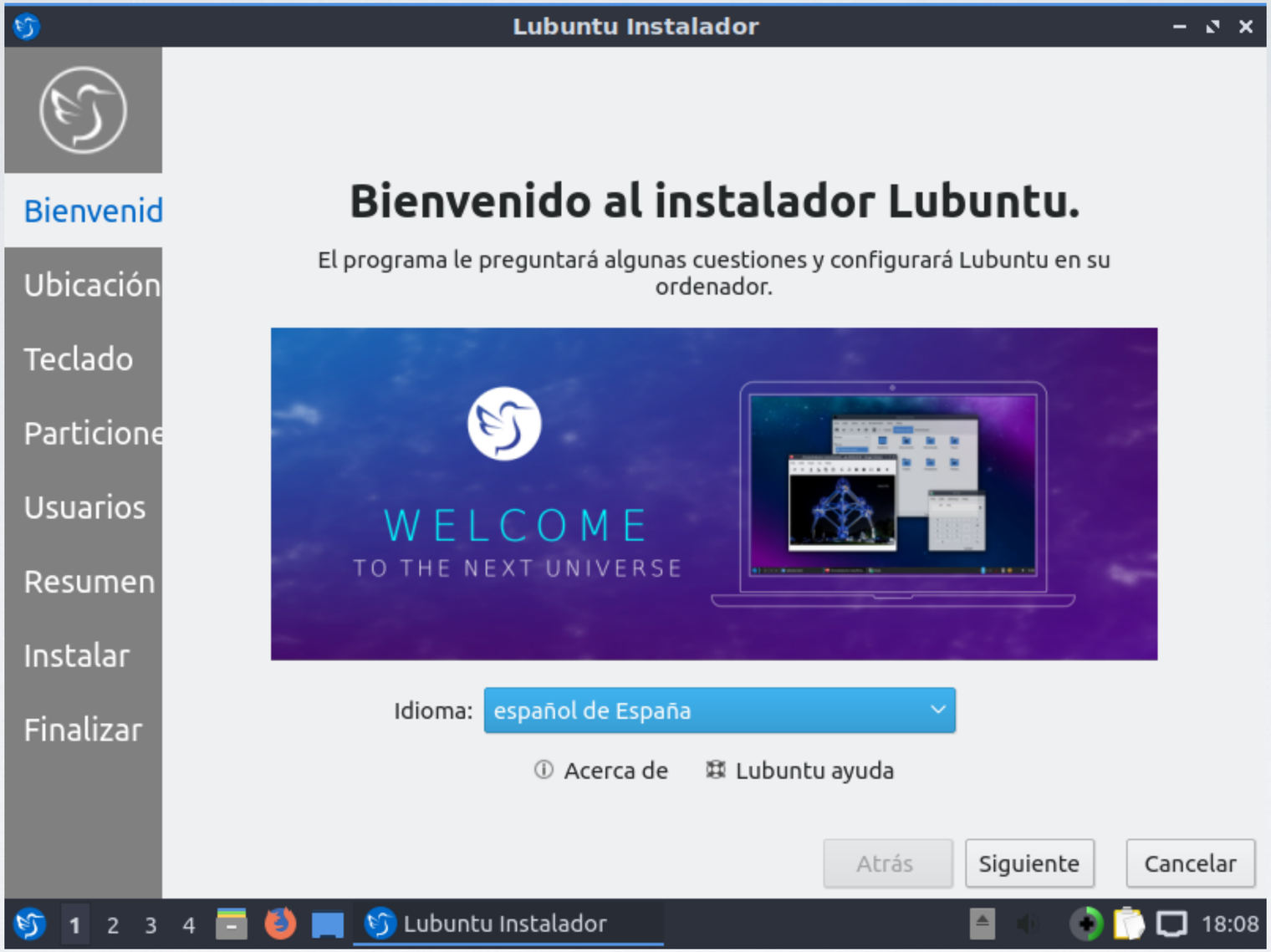


Autor: Radice Adrian

Autor: Radice Adrian

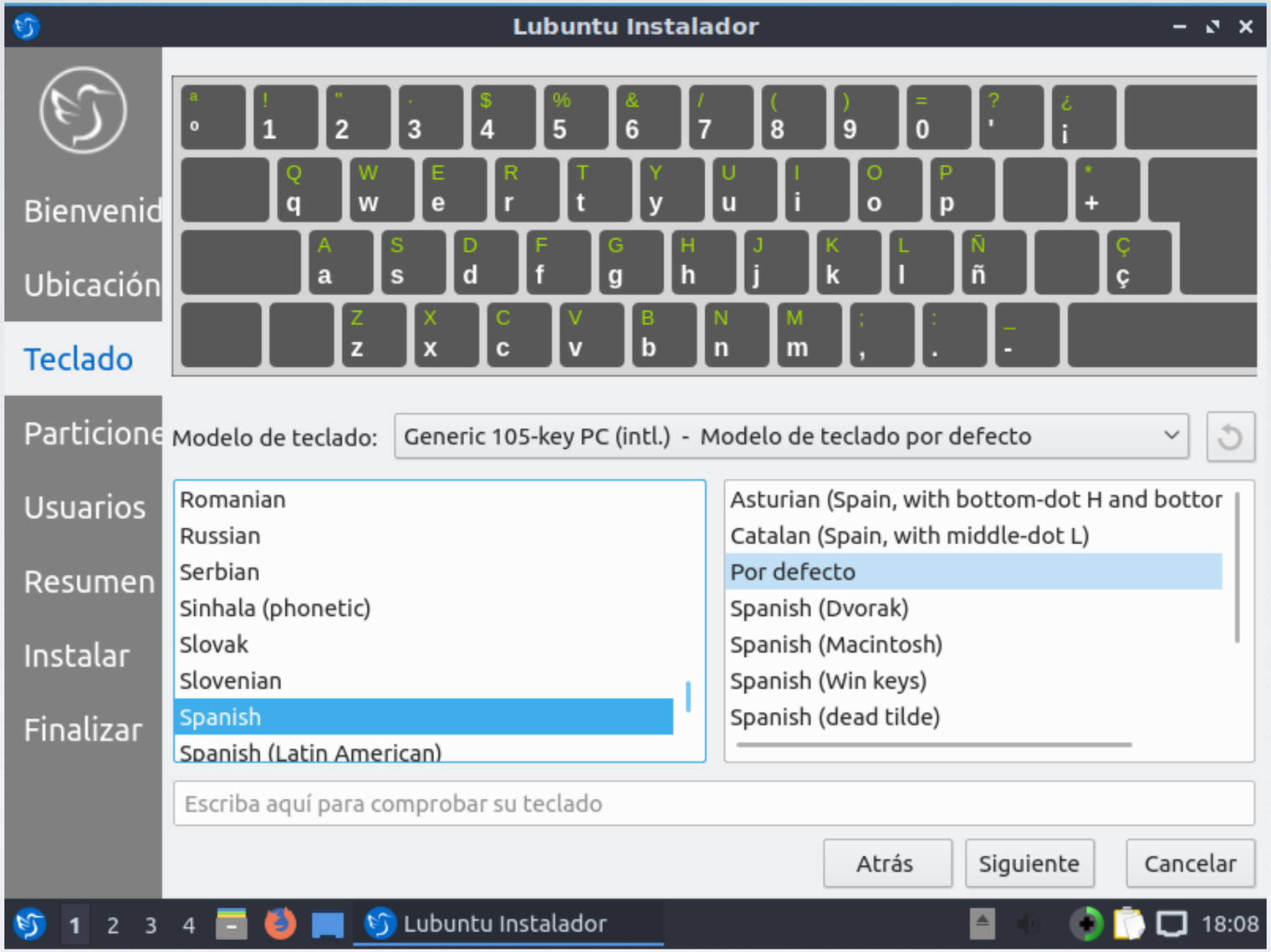


Autor: Radice Adrian

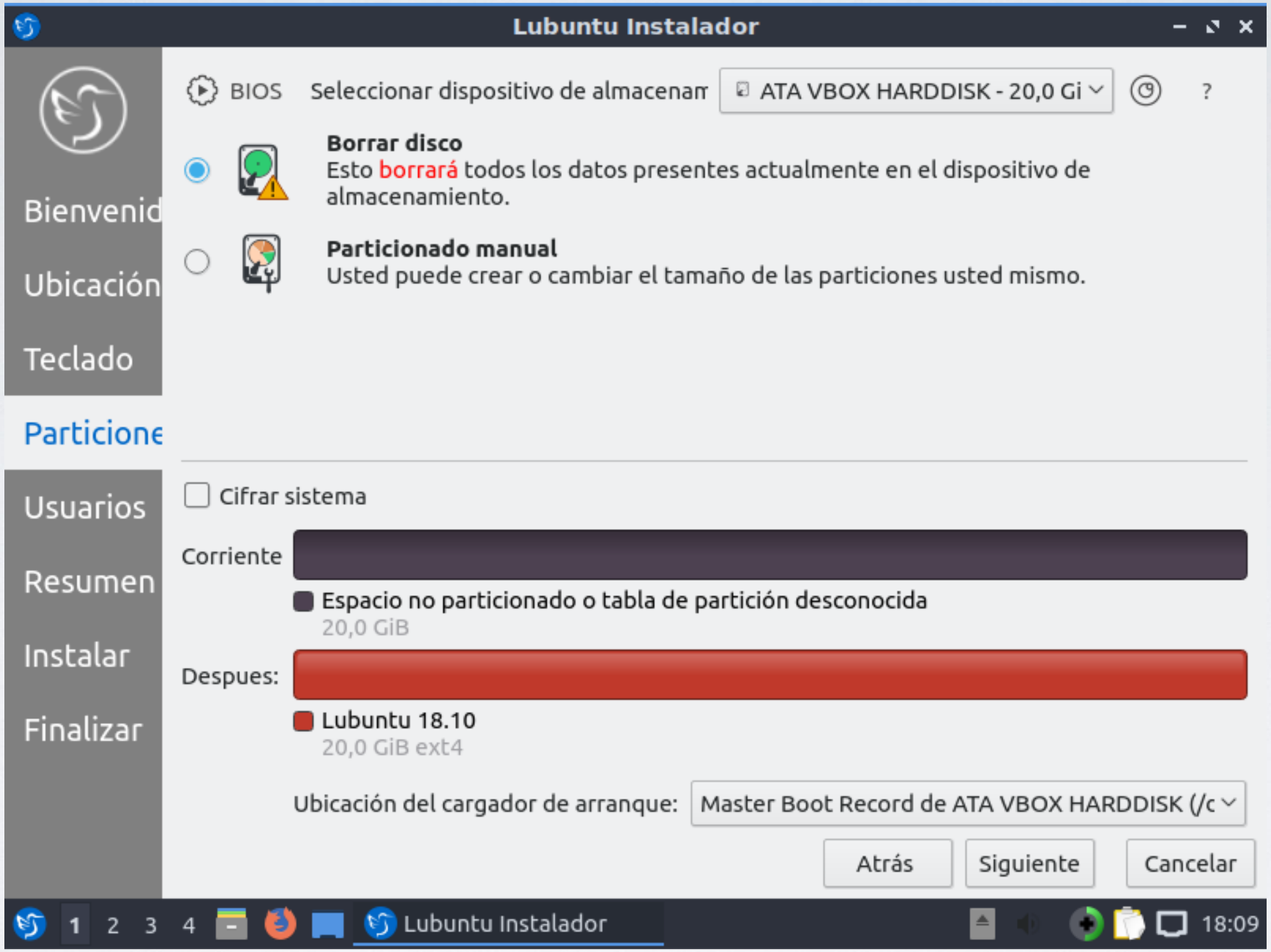


Autor: Radice Adrian

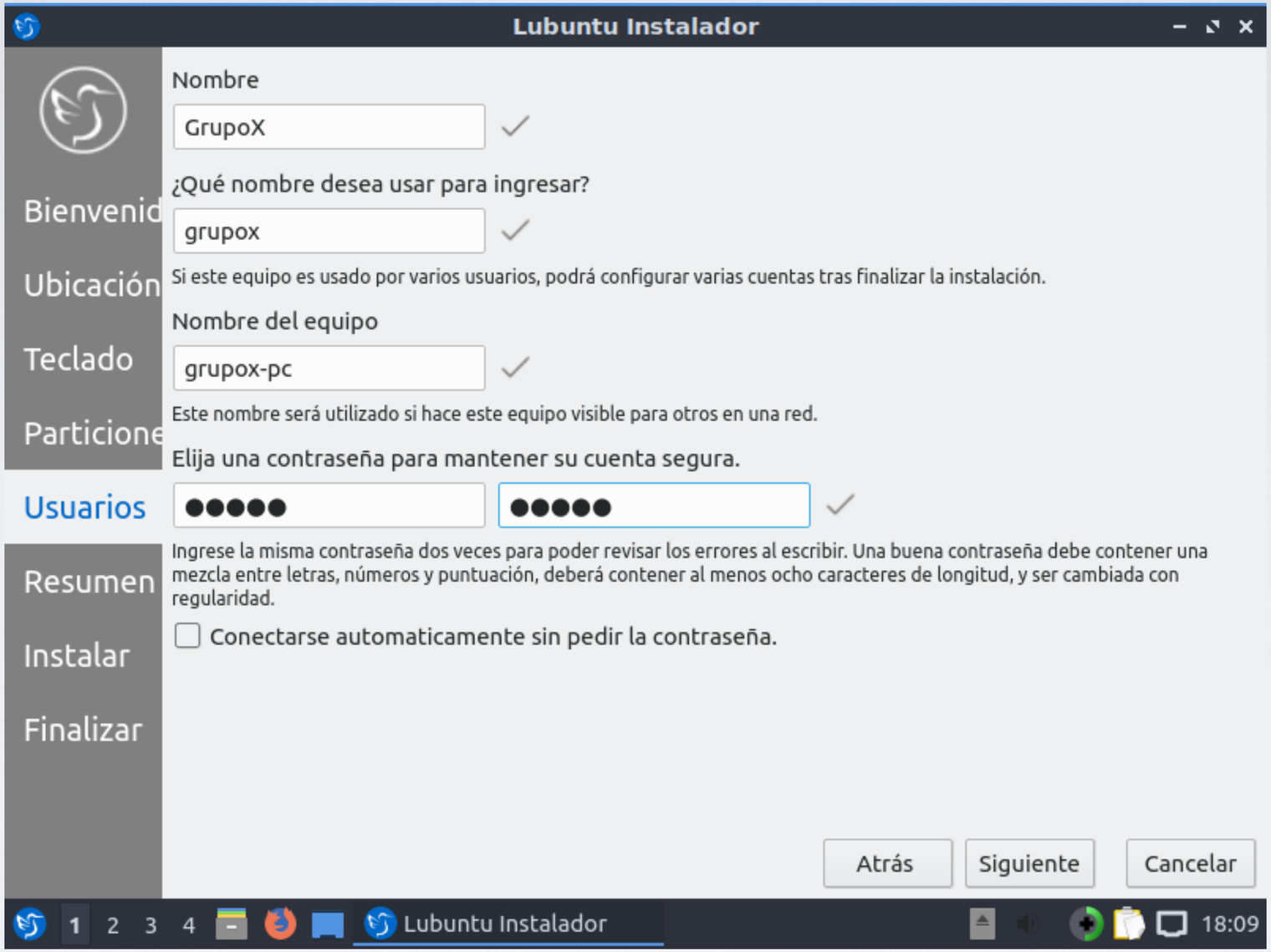
Autor: Radice Adrian



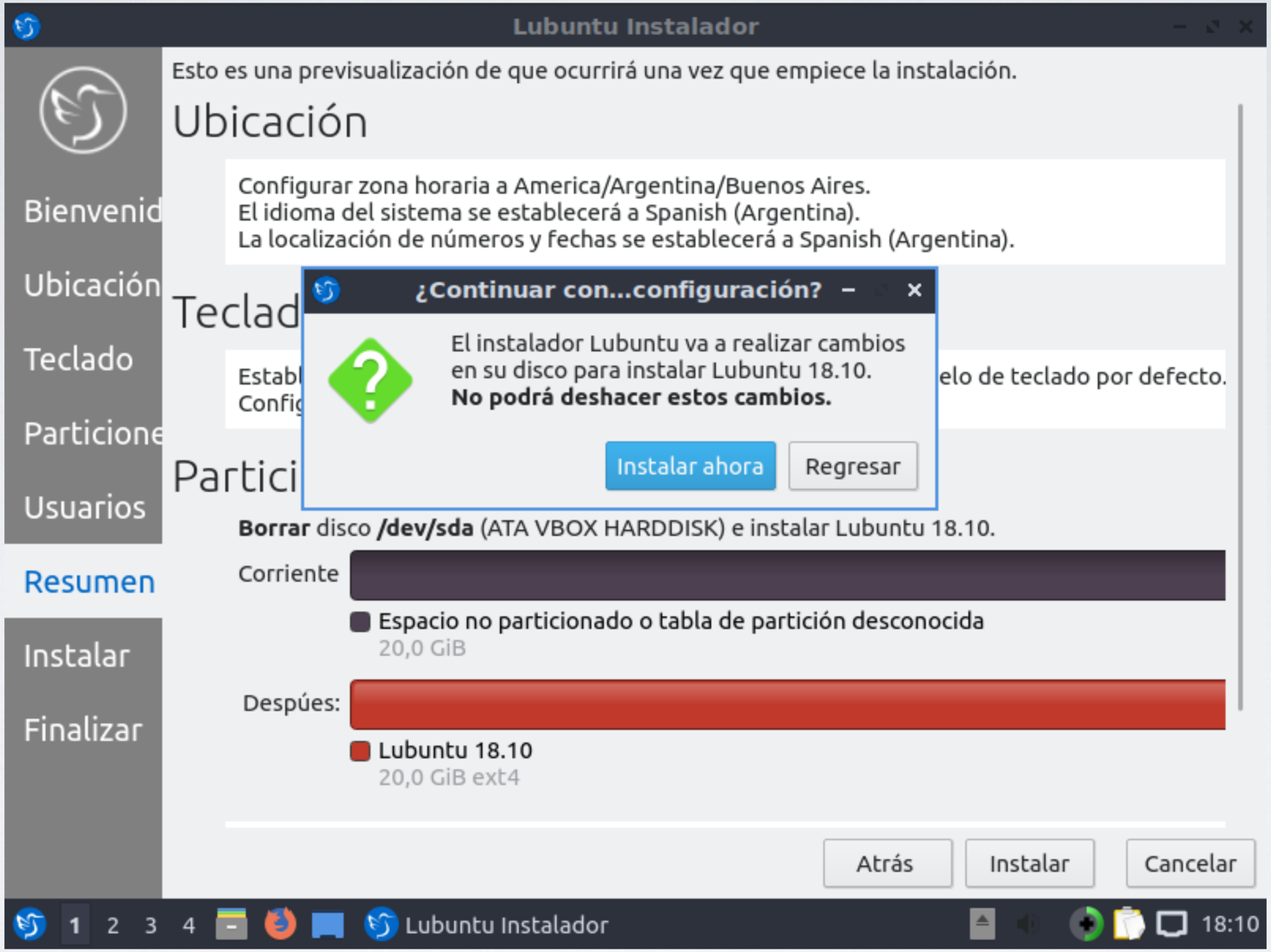
Autor: Radice Adrian



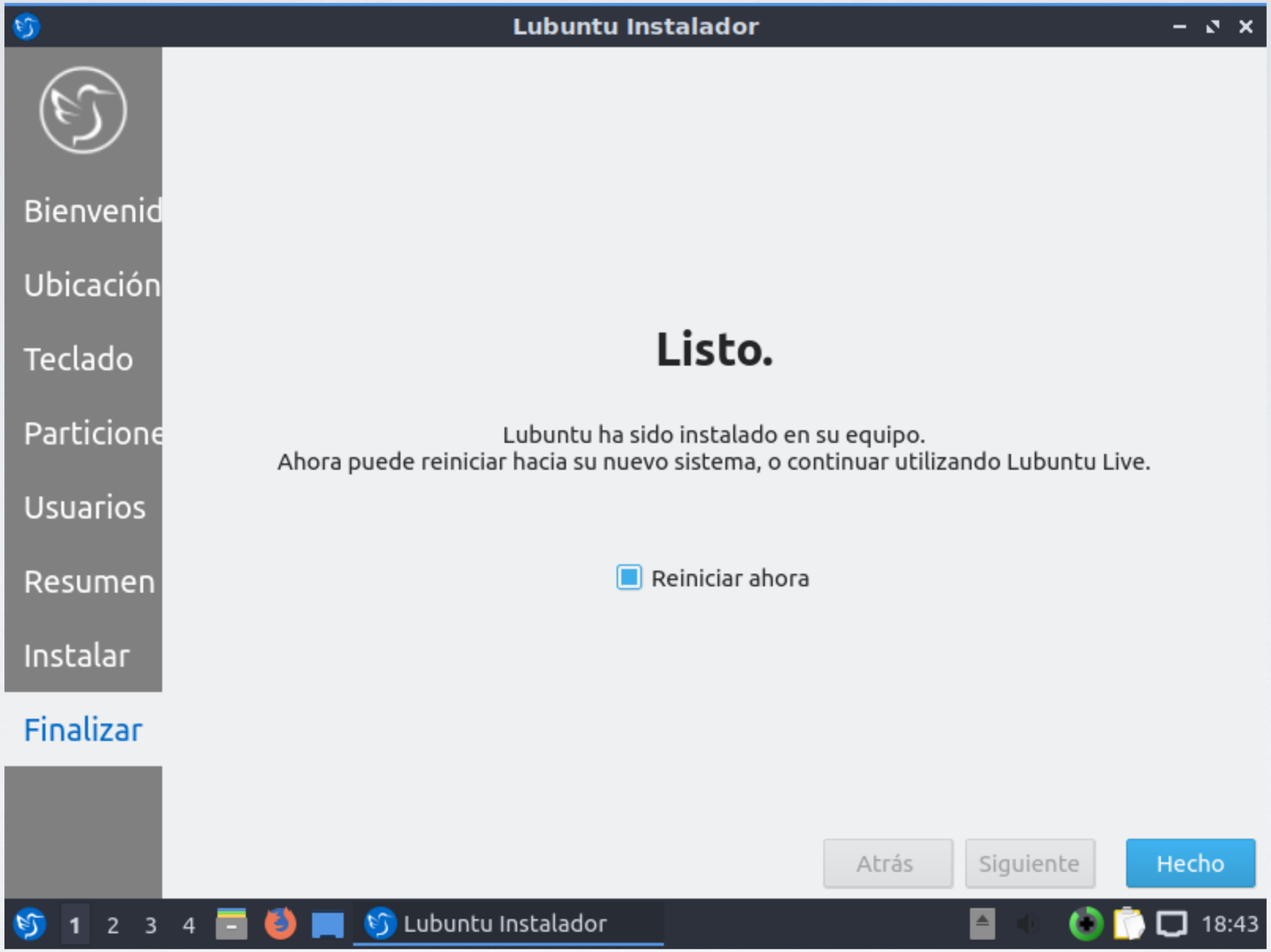
Autor: Radice Adrian



Autor: Radice Adrian

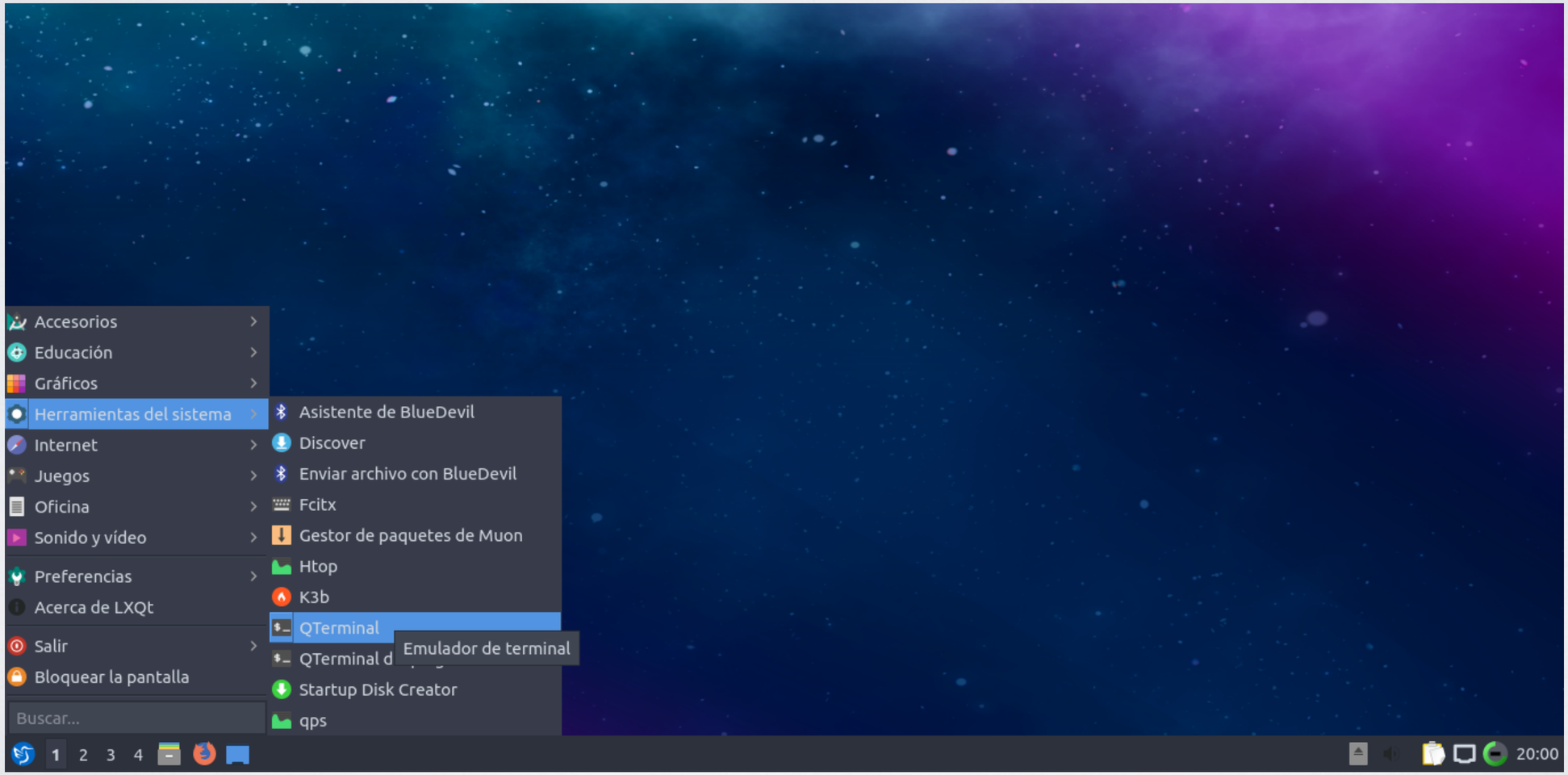


Autor: Radice Adrian



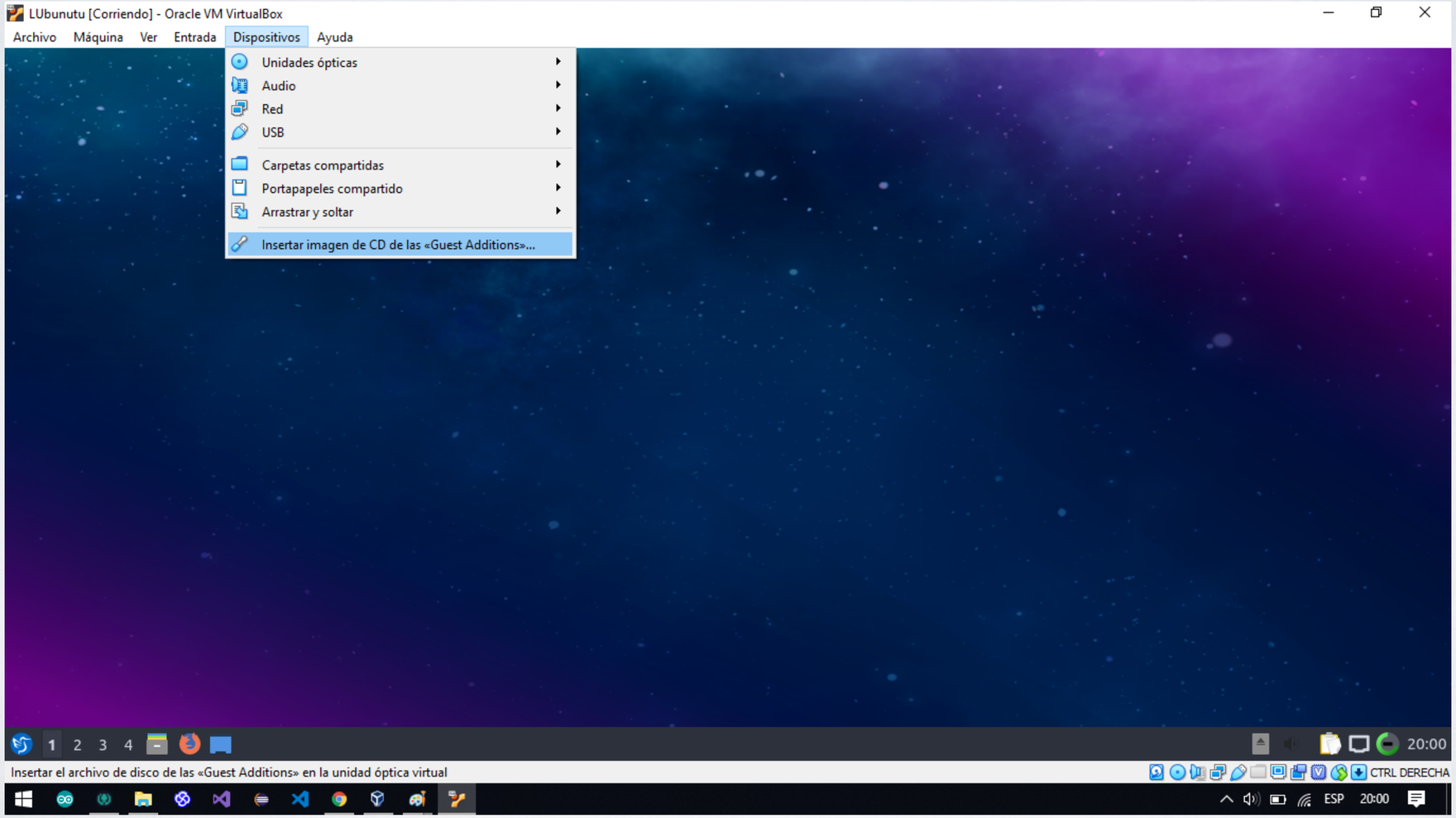
Autor: Radice Adrian

LISTO YA TENEMOS UNA VM CON OS



Autor: Radice Adrian

PARA COMPLETAR LA CONFIG DE LA VM



Autor: Radice Adrian

ABRIMOS UNA TERMINAL Y PEGAMOS EL SIGUIENTE CÓDIGO

```
sudo apt-get update
sudo apt-get upgrade
sudo apt-get install dkms
sudo /media/<nombre_equipo>/VBox_GAs_6.0.4/VBoxLinuxAdditions.run
```

LISTO YA TENEMOS TODO PARA EMPEZAR A TRABAJAR

INTRODUCCIÓN A BASH

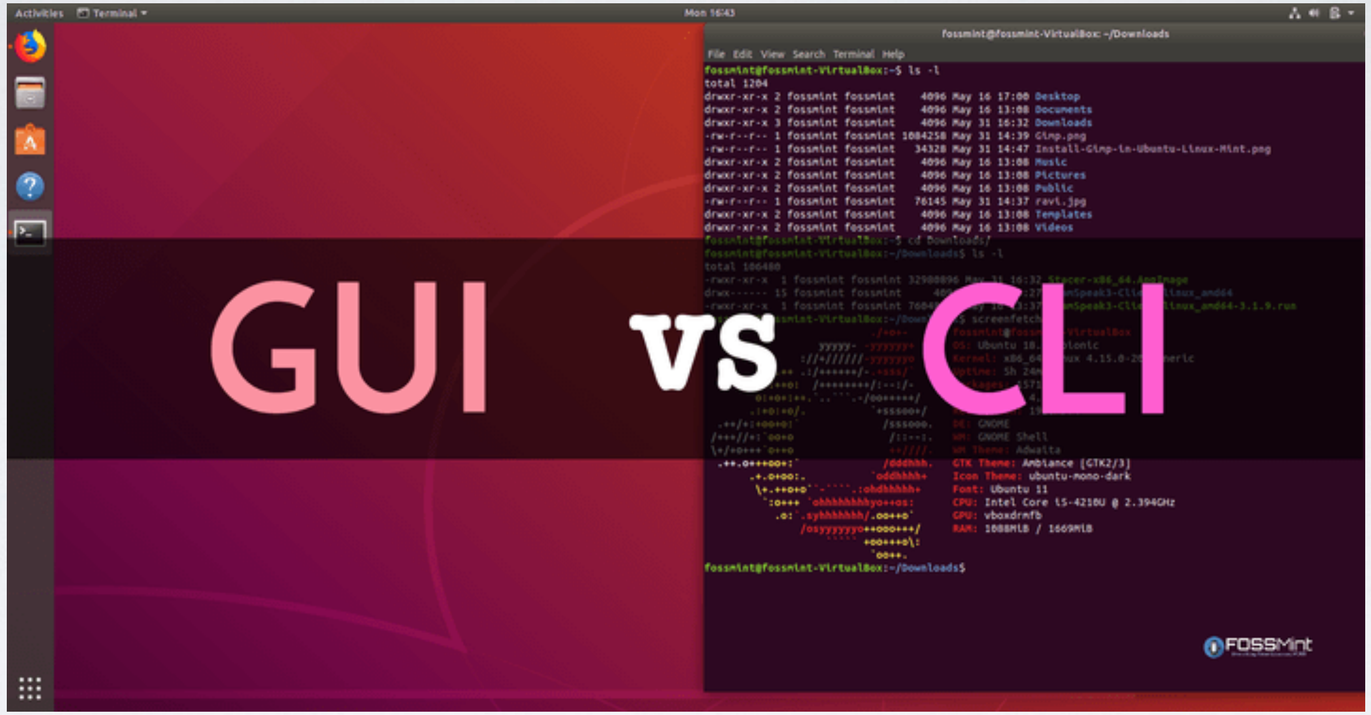
Autor: Radice Adrian

¿QUÉ ES BASH?

¿QUE ES BASH?

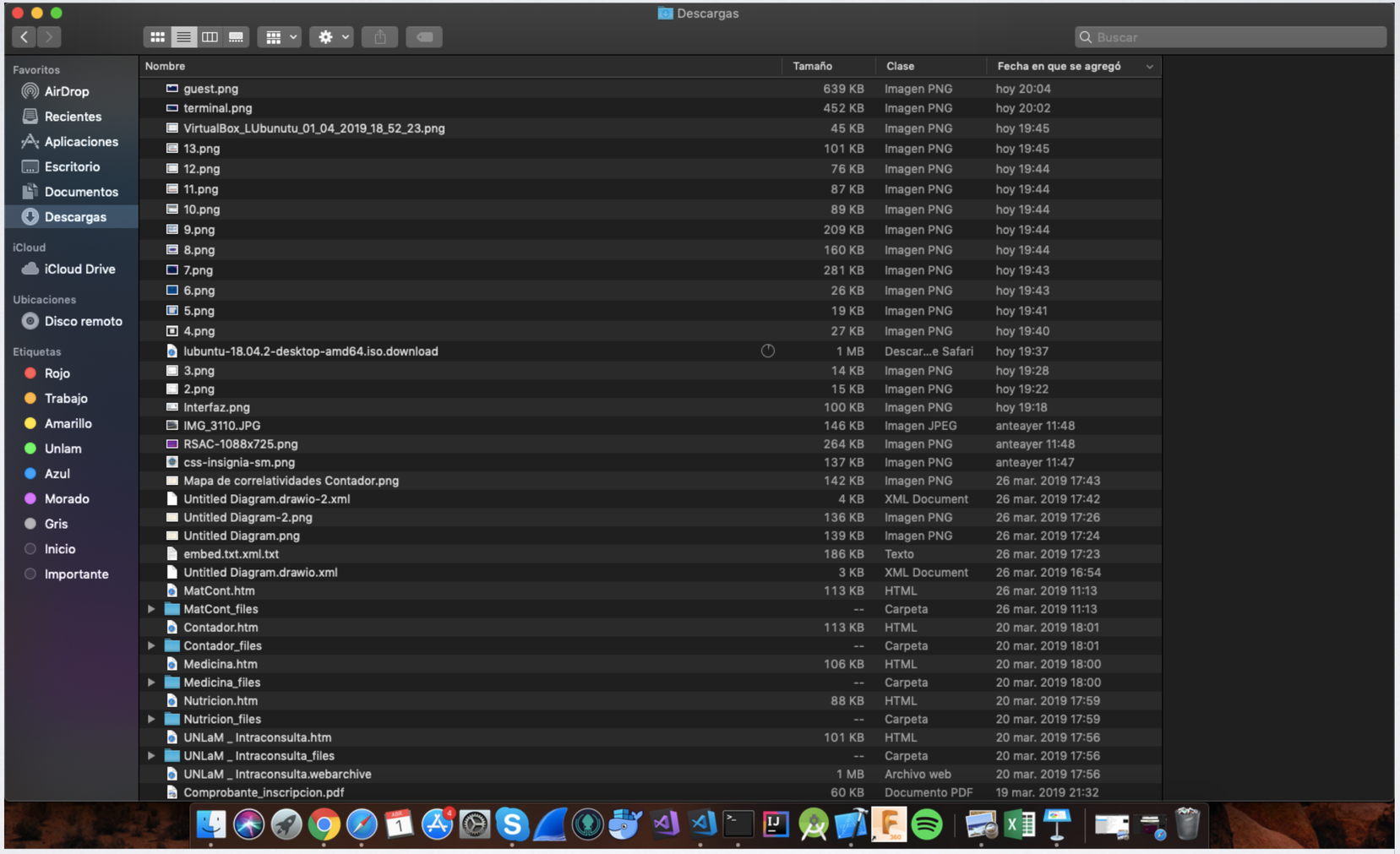
- Es el Interprete de Comando (shell) generalmente por defecto en las distribuciones GNU/Linux.
- Bash a igual que cualquier otro tipo de shell (Ejemplo sh) funciona como interfaz entre el kernel y los usuarios.
- Al interactuar con Bash utilizamos un lenguaje dinámico, orientado a texto.

¿COMO ES LA INTERACCION ENTRE EL USUARIO Y EL SISTEMA OPERATIVO?



Autor: Radice Adrian

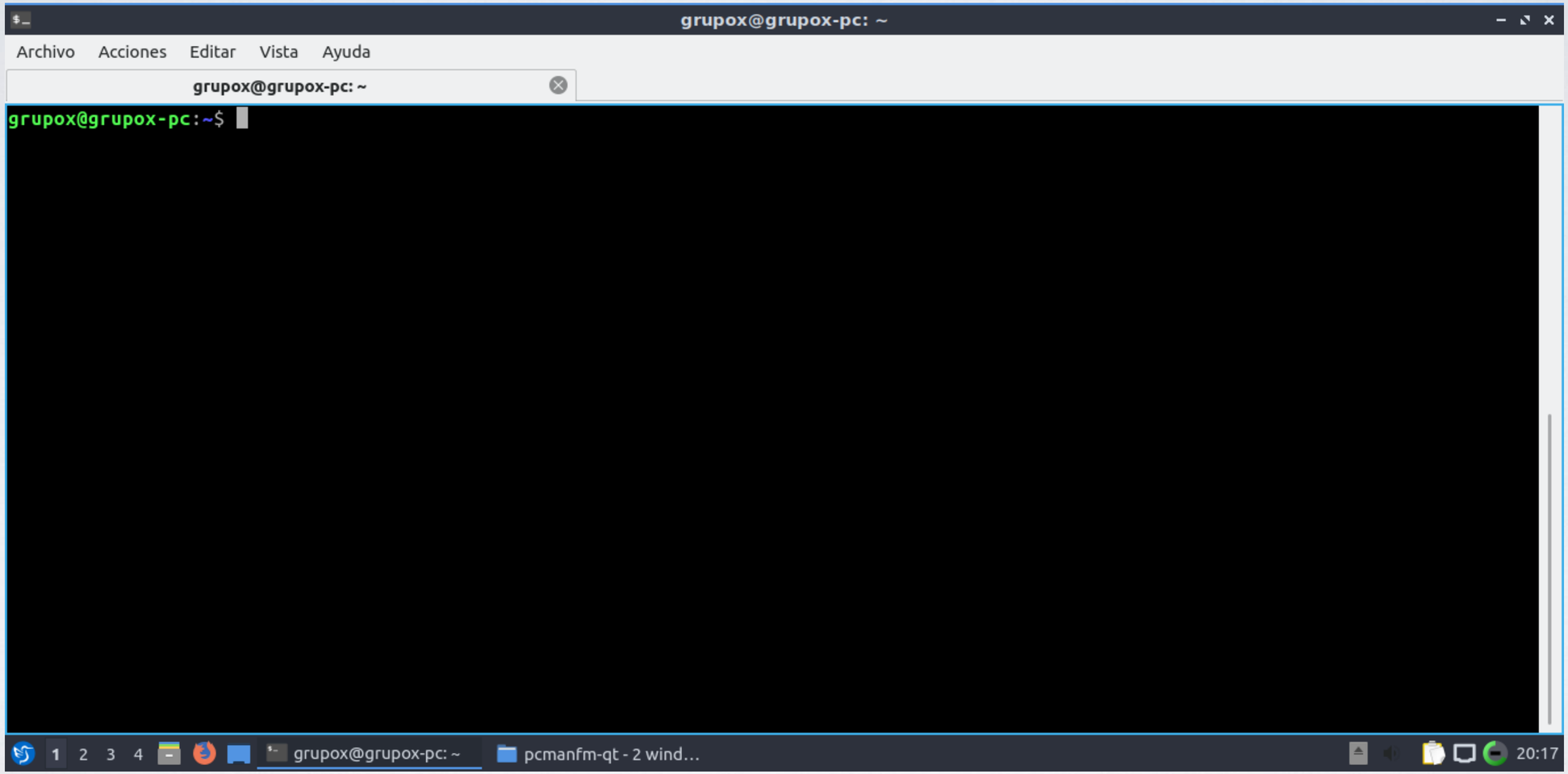
GUI


Descargas
Buscar
Favoritos
AirDrop
Recientes
Aplicaciones
Escritorio
Documentos
Descargas
iCloud
iCloud Drive
Ubicaciones
Disco remoto
Etiquetas
Rojo
Trabajo
Amarillo
Unlam
Azul
Morado
Gris
Inicio
Importante
Nombre
Tamaño
Clase
Fecha en que se agregó
guest.png 639 KB Imagen PNG hoy 20:04
terminal.png 452 KB Imagen PNG hoy 20:02
VirtualBox_LUbunutu_01_04_2019_18_52_23.png 45 KB Imagen PNG hoy 19:45
13.png 101 KB Imagen PNG hoy 19:45
12.png 76 KB Imagen PNG hoy 19:44
11.png 87 KB Imagen PNG hoy 19:44
10.png 89 KB Imagen PNG hoy 19:44
9.png 209 KB Imagen PNG hoy 19:44
8.png 160 KB Imagen PNG hoy 19:44
7.png 281 KB Imagen PNG hoy 19:43
6.png 26 KB Imagen PNG hoy 19:43
5.png 19 KB Imagen PNG hoy 19:41
4.png 27 KB Imagen PNG hoy 19:40
lubuntu-18.04.2-desktop-amd64.iso.download 1 MB Descar...e Safari hoy 19:37
3.png 14 KB Imagen PNG hoy 19:28
2.png 15 KB Imagen PNG hoy 19:22
Interfaz.png 100 KB Imagen PNG hoy 19:18
IMG_3110.JPG 146 KB Imagen JPEG anteayer 11:48
RSAC-1088x725.png 264 KB Imagen PNG anteayer 11:48
css-insignia-sm.png 137 KB Imagen PNG anteayer 11:47
Mapa de correlatividades Contador.png 142 KB Imagen PNG 26 mar. 2019 17:43
Untitled Diagram.drawio-2.xml 4 KB XML Document 26 mar. 2019 17:42
Untitled Diagram-2.png 136 KB Imagen PNG 26 mar. 2019 17:26
Untitled Diagram.png 139 KB Imagen PNG 26 mar. 2019 17:24
embed.txt.xml.txt 186 KB Texto 26 mar. 2019 17:23
Untitled Diagram.drawio.xml 3 KB XML Document 26 mar. 2019 16:54
MatCont.htm 113 KB HTML 26 mar. 2019 11:13
MatCont_files -- Carpeta 26 mar. 2019 11:13
Contador.htm 113 KB HTML 20 mar. 2019 18:01
Contador_files -- Carpeta 20 mar. 2019 18:01
Medicina.htm 106 KB HTML 20 mar. 2019 18:00
Medicina_files -- Carpeta 20 mar. 2019 18:00
Nutricion.htm 88 KB HTML 20 mar. 2019 17:59
Nutricion_files -- Carpeta 20 mar. 2019 17:59
UNLaM _ Intraconsulta.htm 101 KB HTML 20 mar. 2019 17:56
UNLaM _ Intraconsulta_files -- Carpeta 20 mar. 2019 17:56
UNLaM _ Intraconsulta.webarchive 1 MB Archivo web 20 mar. 2019 17:56
Comprobante_inscripcion.pdf 60 KB Documento PDF 19 mar. 2019 21:32

CLI

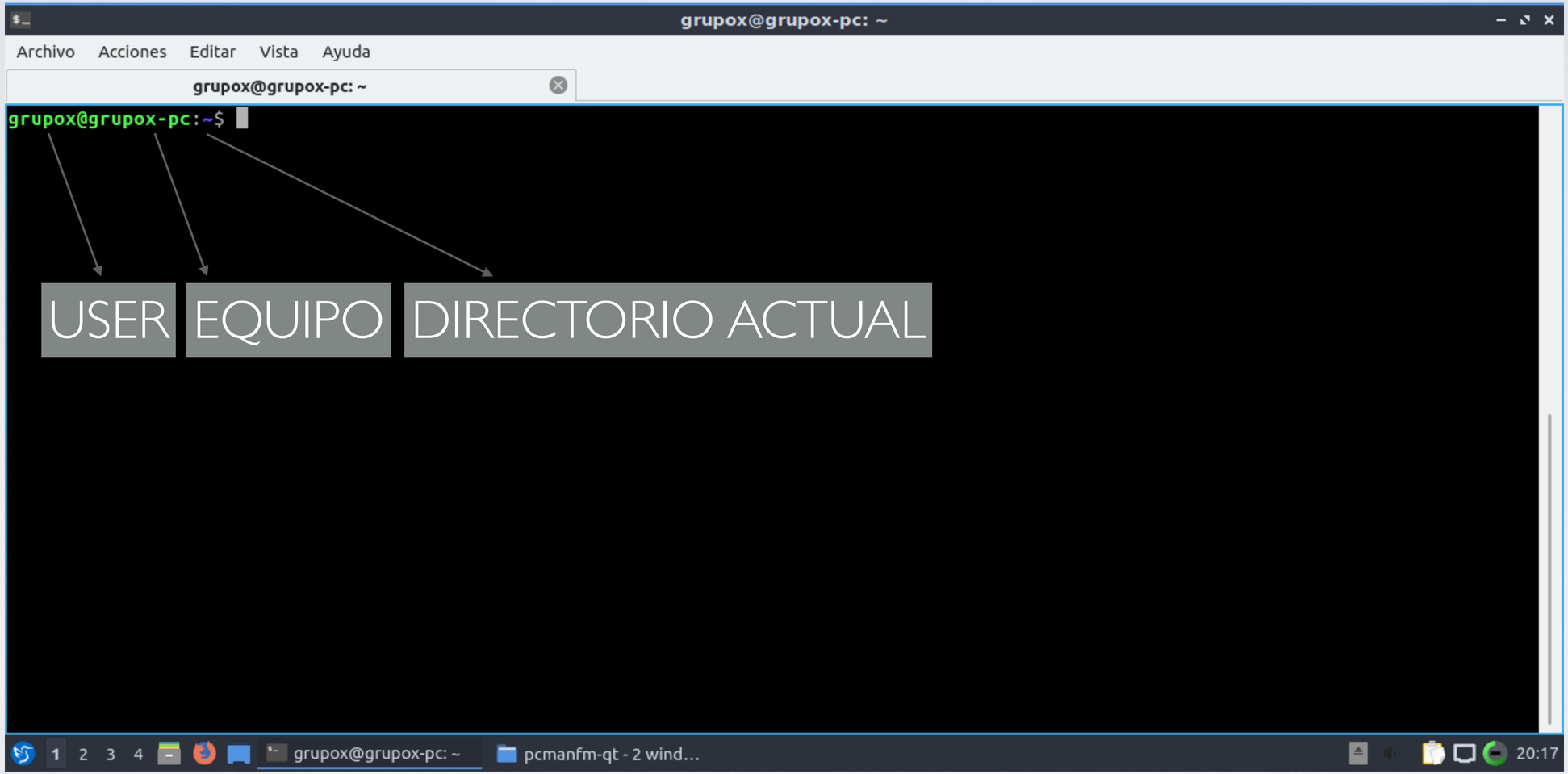
```
Last login: Mon Apr  1 17:30:29 on ttys000
adrianradice:~ 👉 ls
Android Libs          Desktop               IdeaProjects          Los Laureles 1.00     Pictures              VirtualBox VMs        eclipse               nano.save
AndroidStudioProjects Documents             Justinmind            Movies                Projects              agus                  eclipse-workspace     node_modules
Applications          Downloads             Library               Music                 Public                basura                iCloud Drive (archivo)
adrianradice:~ 👉
```

TERMINAL



Autor: Radice Adrian

PROMT



Autor: Radice Adrian

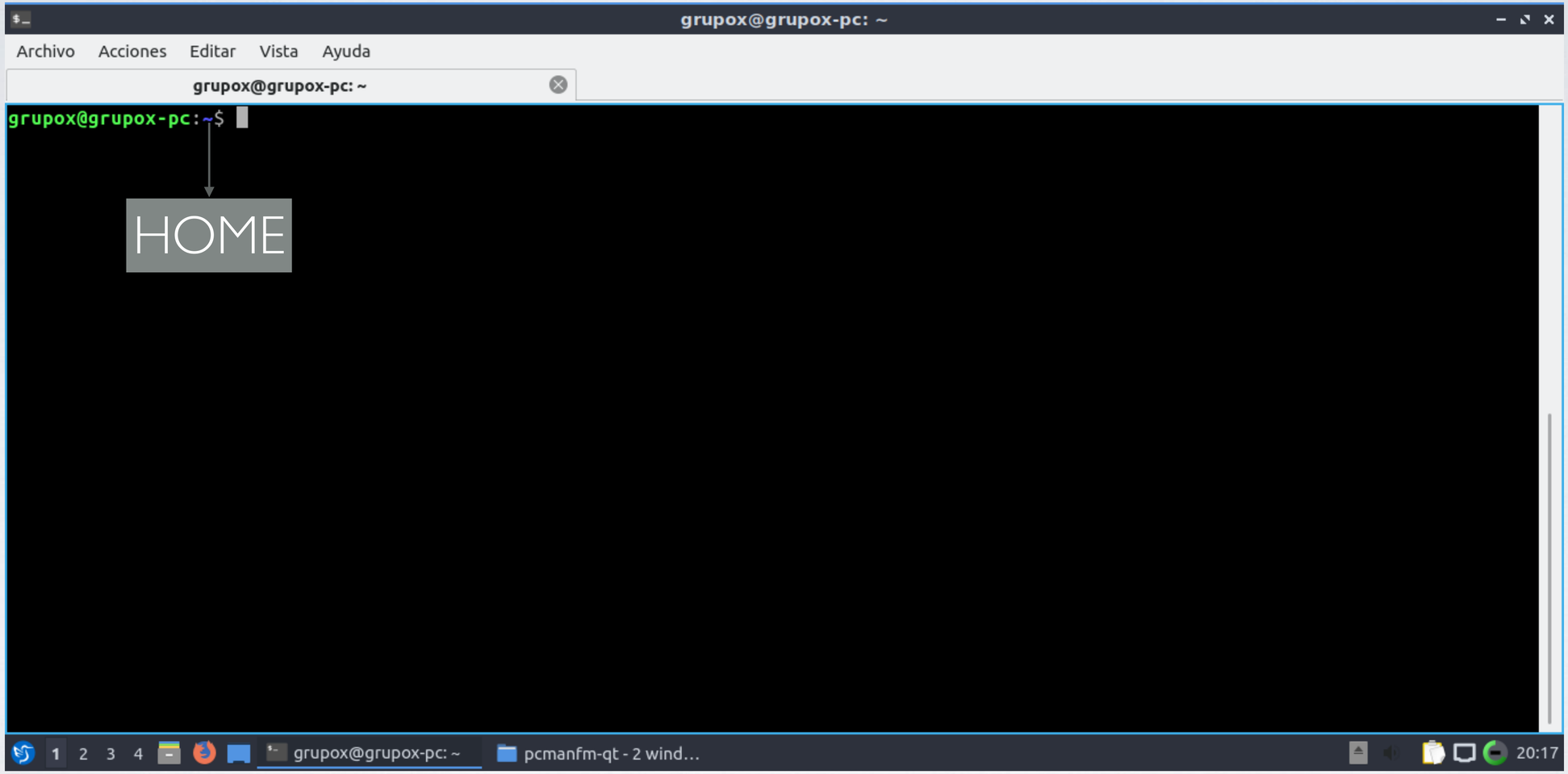
EMPEZEMOS A JUGAR UN POCO

LO PRIMERO QUE DEBERÍAMOS CONOCER ES LA ESTRUCTURA DE DIRECTORIOS DESDE LA RAÍZ

EL DIRECTORIO RAÍZ EN LINUX ES /

/ es equivalente a C:/ de windows por ejemplo

PERO NUESTRO PROMT ACTUALMENTE INDICA QUE ESTAMOS EN HOME



HOME es equivalente a la carpeta del usuario en windows

Autor: Radice Adrian

ESTAMOS EN HOME Y QUEREMOS IR A /

OSEA QUEREMOS CAMBIAR DE DIRECTORIO

CHANGE **D**IRECTORY

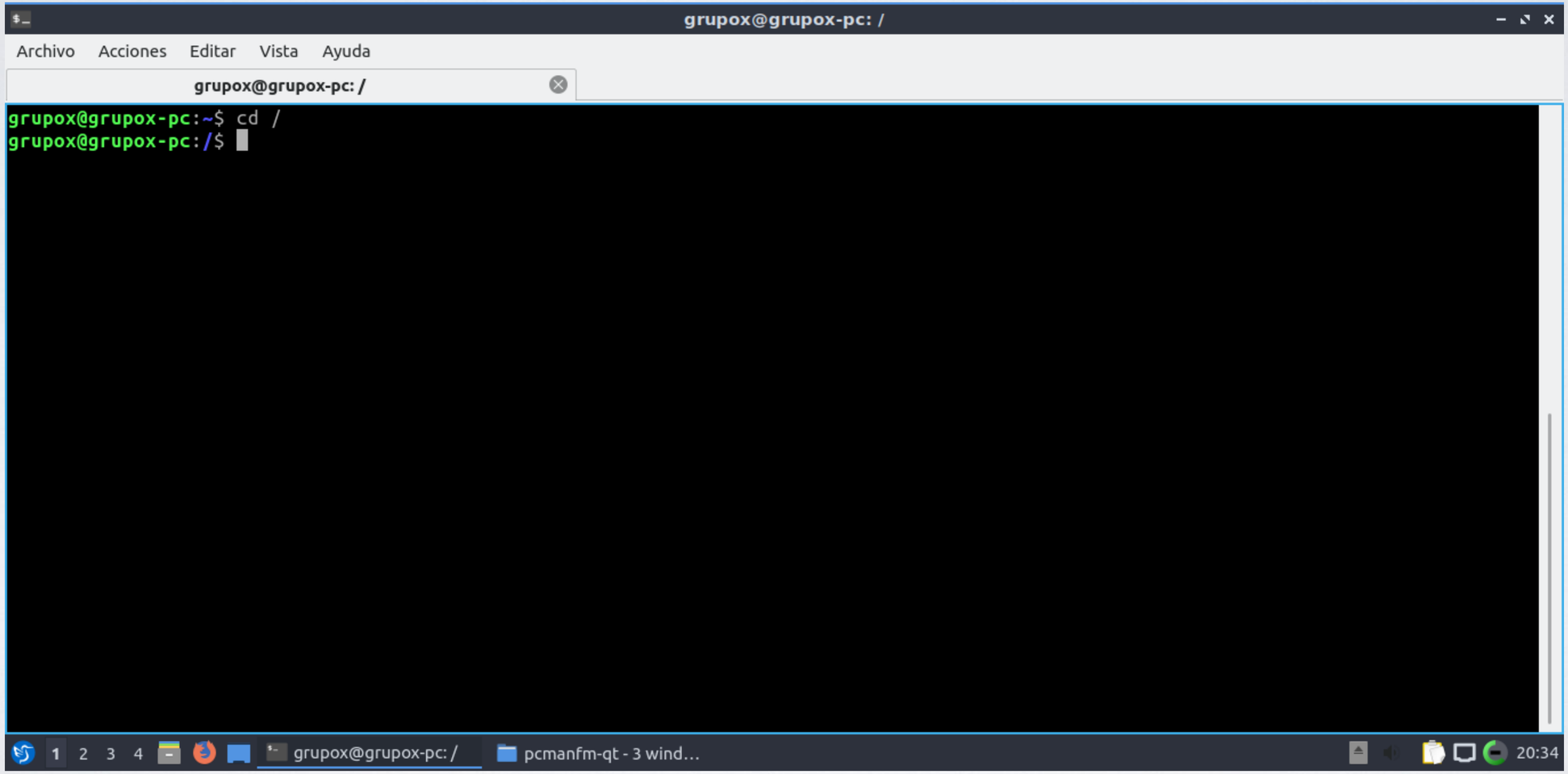
CAMBIAR DIRECTORIO

Autor: Radice Adrian

COMANDO CD

CD <DESTINO>

cd /



LISTO LLEGAMOS A LA RAÍZ

¿Y ahora cómo veo la estructura?

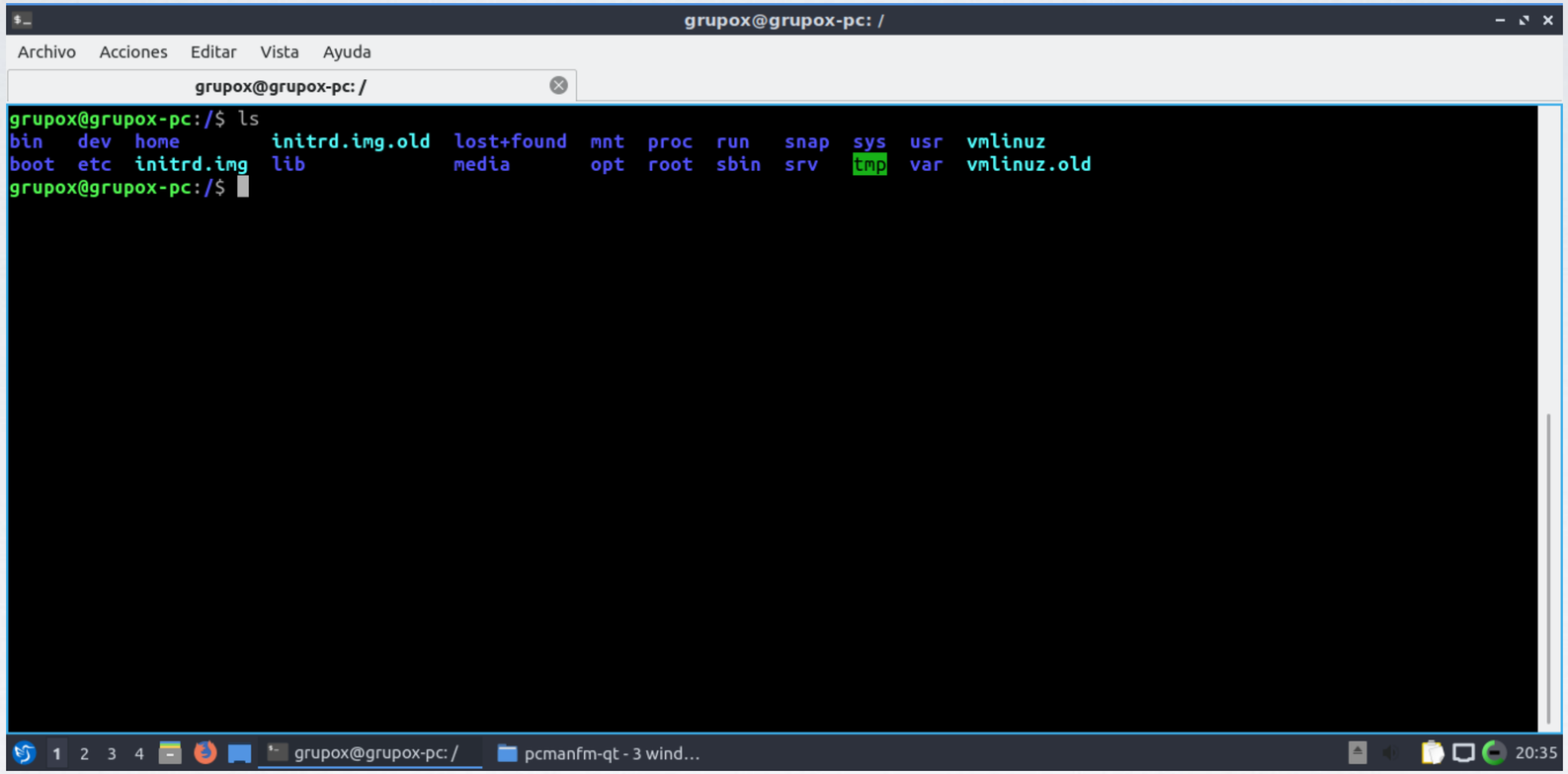
RECORDEMOS QUE EN LINUX UN DIRECTORIO ES UN ARCHIVO

ENTONCES UN DIRECTORIO SERÍA COMO UN ‘“”’’’’’LISTADO DE NOMBRES DE OBJETOS GUARDADOS’’’’’’’’

ENTONCES DEBERIAMOS LISTAR EL CONTENIDO DEL DIRECTORIO

LISTAR <==> LIST

COMANDO LS



Autor: Radice Adrian

ALGUNOS DIRECTORIOS

/ (raíz) Es el nivel más alto dentro de la jerarquía de directorios.
De aquí cuelgan el resto de carpetas, particiones y otros dispositivos.

/bin (binarios) Los binarios son los ejecutables de Linux, aquí se encontrar los comandos propios del Sistema Operativo. Por ejemplo: cp, mv, cat

/boot (arranque) Aquí se encuentra los archivos necesarios para el inicio del sistema

/temp (temporal) Es un directorio donde se almacenan fi cheros temporales.
Cada vez que se inicia el sistema este directorio se limpia.

/home (hogar) Este hogar no es más que un directorio que a su vez contiene otros, uno por cada usuario dado de alta en el sistema

ALGUNOS DIRECTORIOS

/root (hogar admin) Es el /home del administrador. Es el único /home que no está incluido -por defecto- en el directorio anteriormente mencionado.

/etc (etcetera) Aquí se guardan los ficheros de configuración de los programas instalados, así como ciertos scripts que se ejecutan en el inicio del sistema. (pueden complementarse o sustituirse por los por los ficheros que cada user tiene en su home)

/media (medios) Es donde se montan las unidades extraíbles como los dispositivos USB, disqueteras, unidades de CD/DVD y en algunas distros, como Ubuntu, las particiones adicionales

/proc (process) Información de los procesos

SIGAMOS JUGANDO

AHORA CREEMOS EL DIRECTORIO PARA NUESTROS TP'S

¿CÓMO ERA CREAR EN INGLES?

Ma**K**e

¿CÓMO SE DICE DIRECTORIO EN INGLES?

Directory

MA**K**E + **DIR**ECTORY

mkdir

mkdir <path>
Crea un directorio en el path indicado y con el nombre correspondiente

```
[adrianradice:Desktop 👉 mkdir Sisop
[adrianradice:Desktop 👉 cd Sisop
[adrianradice:Sisop 👉 mkdir Practicos
[adrianradice:Sisop 👉 cd Practicos/
[adrianradice:Practicos 👉 mkdir bash
```

AHORA CREEMOS EL ARCHIVO QUE CONTENDRÁ NUESTRO PRIMER SCRIPT

PARA CREAR UN ARCHIVO TENEMOS MUCHAS FORMAS

Probemos una!

COMANDO TOUCH

Este comando permite crear un archivo nuevo

```
[adrianradice:bash 👉 touch source.sh
[adrianradice:bash 👉 ls
 EJ1                source.sh
```

NOTA: Si ejecutamos touch y como parámetro pasamos el nombre de un archivo que existe, solamente cambia la fecha de modificación por la actual, NO CAMBIA EL CONTENIDO DEL MISMO

UPS! PERO NOS QUEDO MAL UBICADO EL ARCHIVO, DEBERÍA ESTAR EN EJ1

Y AHORA QUE HACEMOS?

TENEMOS QUE MOVER EL ARCHIVO

MOVER…… MOVE….

COMANDO MV

mv <origen> <destino>

Mueve el archivo de la ubicación actual a la ubicación destino

```
[adrianradice:bash 👉 ls
EJ1             source.sh
[adrianradice:bash 👉 mv source.sh EJ1/source.sh
[adrianradice:bash 👉 ls
EJ1
[adrianradice:bash 👉 cd EJ1/
[adrianradice:EJ1 👉 ls
source.sh
adrianradice:EJ1 👉
```

SUPONGAMOS QUE NUESTRO SCRIPT EJECUTA LS

EDITAMOS SOURCE.SH CON NUESTRO EDITOR FAVORITO

EJEMPLO VI

```
adrianradice:EJ1 👉 vi source.sh
```

AHORA EJECUTAMOS EL SCRIPT SOLAMENTE MENCIONÁNDOLO

```
adrianradice:EJ1 👉 ./source.sh
-bash: ./source.sh: Permission denied
```

UPS! NO TENEMOS PERMISOS

Y AHORA?

Que permisos tiene esto que creamos?

LOS PERMISOS SON INFORMACIÓN DE LOS ARCHIVOS Y PODEMOS VERLOS DESDE SU DIRECTORIO

Entonces vuelve el comando ls al rescate

PERO LS SOLO NO NOS SIRVE
NECESITAMOS ESPECIFICARLE
QUE QUEREMOS MAS INFO

SI QUEREMOS ESPECIFICAR ALGO QUE NECESITAMOS

Parametros

Autor: Radice Adrian

COMO OBTENEMOS INFO SOBRE LOS COMANDOS EXTERNOS. CONSULTANDO EL MANUAL

man <nombre comando>

EJECUTAMOS POR EJEMPLO MAN LS

```
LS(1)                    BSD General Commands Manual                    LS(1)

NAME
     ls -- list directory contents

SYNOPSIS
     ls [-ABCFGHLOPRSTUW@abcdefghiklmnopqrstuwx1] [file ...]

DESCRIPTION
     For each operand that names a file of a type other than directory, ls displays its name as well as any requested, associated information.
     For each operand that names a file of type directory, ls displays the names of files contained within that directory, as well as any
     requested, associated information.

     If no operands are given, the contents of the current directory are displayed.  If more than one operand is given, non-directory operands
     are displayed first; directory and non-directory operands are sorted separately and in lexicographical order.

     The following options are available:

     -@      Display extended attribute keys and sizes in long (-l) output.

     -1      (The numeric digit ``one''.)  Force output to be one entry per line.  This is the default when output is not to a terminal.

     -A      List all entries except for . and ..  Always set for the super-user.

     -a      Include directory entries whose names begin with a dot (.).

     -B      Force printing of non-printable characters (as defined by ctype(3) and current locale settings) in file names as \xxx, where xxx
             is the numeric value of the character in octal.

     -b      As -B, but use C escape codes whenever possible.

     -C      Force multi-column output; this is the default when output is to a terminal.

     -c      Use time when file status was last changed for sorting (-t) or long printing (-l).

     -d      Directories are listed as plain files (not searched recursively).

     -e      Print the Access Control List (ACL) associated with the file, if present, in long (-l) output.

     -F      Display a slash (`/') immediately after each pathname that is a directory, an asterisk (`*') after each that is executable, an at
             sign (`@') after each symbolic link, an equals sign (`=') after each socket, a percent sign (`%') after each whiteout, and a ver-
             tical bar (`|') after each that is a FIFO.

     -f      Output is not sorted.  This option turns on the -a option.

:
```

Autor: Radice Adrian

DE LA AYUDA DETERMINAMOS QUE CON -L VEMOS LOS PERMISOS ENTRE OTRAS COSAS

```
adrianradice:EJ1 👉 ls -l
total 8
-rw-r--r--  1 adrianradice  staff  3 Apr  1 21:19 source.sh
```

También podemos concatenar parámetros por ejemplo
ls -la mostraría en formato de lista (l list) y archivos ocultos (a all)

COMO LEEMOS ESTO?

```
adrianradice:EJ1 👉 ls -l
total 8
-rw-r--r--  1 adrianradice  staff  3 Apr  1 21:19 source.sh
```

PERMISO PARA Otros

PERMISO PARA el GRUPO

Tipo de elemento
l -> Indo
D -> dir
- -> Archivo

PERMISOS PARA EL USUARIO PROPIETARIO

PERMISOS EN LINUX

Los sistemas Linux poseen tres permisos básicos que denotan la capacidad de cierto usuario para <<Leer>> <<Escribir>> o <<Ejecutar>> determinado archivo / directorio

R - Read
W - Write
X - eXecute

A QUIENES SE LE PUEDEN ASIGNAR PERMISOS

U - dueño
G - grupo
O - Otros
A - Todos

Autor: Radice Adrian

COMO EDITAMOS LOS PERMISOS

Con el comando chmod (change mode)

Autor: Radice Adrian

CHMOD TIENE MUCHAS FORMAS DE ACEPTAR SUS PARÁMETROS

Veamos una de ellas

CHMOD -> OCTAL

chmod NNN PATH_FICHERO

U G O

CADA N ES UN NUMERO OCTAL

Número	Binario	Lectura (*r*)	Escritura (*w*)	Ejecución (*x*)
0	000	✗	✗	✗
1	001	✗	✗	✓
2	010	✗	✓	✗
3	011	✗	✓	✓
4	100	✓	✗	✗
5	101	✓	✗	✓
6	110	✓	✓	✗
7	111	✓	✓	✓

NINGUN PERMISO

TODOS LOS PERMISOS

CHMOD Aritmetico

chmod {<U|O|G><=|+|-><R|W|X>,}+ PATH_FICHERO

Ejemplo

Chmod u+x, g=r, o-rwx source.sh seria equivalente a

Chmod 740 source.sh

APLIQUEMOS CHMOD A NUESTRO EJEMPLO

```
adrianradice:EJ1 👉 chmod u+x source.sh
adrianradice:EJ1 👉 ./source.sh
source.sh
```

COMANDO SUDO

Para realizar diversas operaciones nos pedirán permisos de super-usuario o root.

```
[adrianradice:bash 👉 sudo rm -rf EJ2
[Password:
Sorry, try again.
[Password:
[adrianradice:bash 👉 sudo su
root:bash 👉
```

En el primer caso ejecutamos un comando como super user

En El segundo caso cambiamos a una sesión super user hasta que cerremos la terminal o ejecutemos exit

COMANDO STARTX

INICIA LA INTERFAZ GRAFICA EN UNA SESIÓN

Autor: Radice Adrian

ARCHIVOS OCULTOS

Los archivos ocultos en linux se indican por un punto delante de su nombre

CÓMO CREAMOS UN ARCHIVO OCULTO?

```
[adrianradice:bash 👉 ls
 EJ1
[adrianradice:bash 👉 mkdir .EJ2
[adrianradice:bash 👉 ls
 EJ1
[adrianradice:bash 👉 ls -a
.        ..       .EJ2    EJ1
 adrianradice:bash 👉
```

Y SI QUIERO QUE SEA VISIBLE AHORA QUE HAGO?

```
[adrianradice:bash 👉 ls -a
.        ..       .EJ2    EJ1
[adrianradice:bash 👉 mv .EJ2 EJ2
[adrianradice:bash 👉 ls
EJ1     EJ2
adrianradice:bash 👉
```

Notar que mv también puede usarse para cambiar el nombre del archivo, o cambiar su visibilidad, cuando la ruta destino y origen son iguales

SEGURAMENTE TE ESTES PREGUNTANDO QUE ES .Y ..

RUTA RELATIVA

Una ruta relativa hace referencia a una ubicación que es relativa a un directorio actual. Las rutas relativas utilizan dos símbolos especiales, un punto (.) y dos puntos seguidos (..), lo que significa el directorio actual y el directorio padre. Los dos puntos seguidos se utilizan para subir en la jerarquía. Un único punto representa el directorio actual.

. Es directorio actual

.. Es el directorio anterior

NOTA: Si las rutas contienen espacios en sus nombres no olvide siempre encerrar
Ejemplo
path="prueba dir con espacios"
Mkdir "$path" de esta forma aseguramos que no interfiera el IFS y admita las ruta

ANALIZAR

```
[adrianradice:bash 👉 ls
EJ1     EJ2
[adrianradice:bash 👉 ls EJ1
source.sh       test
[adrianradice:bash 👉 ls EJ1/test
source2.sh
[adrianradice:bash 👉 cd EJ1/test
[adrianradice:test 👉 mv source2.sh ../../EJ2
[adrianradice:test 👉 cd ..
[adrianradice:EJ1 👉 cd ..
[adrianradice:bash 👉 ls
EJ1     EJ2
[adrianradice:bash 👉 ls EJ2
source2.sh
adrianradice:bash 👉
```

OTROS COMANDO

cp copy
cat concatenar
rm remover
clear limpiar pantalla
echo imprimir por pantalla
read leer del teclado
ps mostrar los procesos actuales en el sistema
pwd mostrar la ruta actual

```
[adrianradice:bash 👉 ls
EJ1
[adrianradice:bash 👉 cp -r EJ1 EJ2
[adrianradice:bash 👉 ls
EJ1     EJ2
[adrianradice:bash 👉 rm EJ2/source.sh
[adrianradice:bash 👉 cd EJ2
[adrianradice:EJ2 👉 ls
[adrianradice:EJ2 👉 cd ..
[adrianradice:bash 👉 ls EJ1
source.sh
[adrianradice:bash 👉 cat EJ1/source.sh
ls
adrianradice:bash 👉
```

El parámetro -r indica que operamos con directorios

EXTENSIONES

A diferencia de otros sistemas operativos, Linux no utiliza las extensiones de los archivos para determinar su tipo.

SI NO HAY EXTENSIONES
COMO SABE EL TRATAMIENTO
QUE TIENE QUE APLICAR EL OS?

MAGIC NUMBER

Son los primeros bit de un archivo y indican el tipo de archivo. El valor de estos bit esta definido por un estándar ISO, aunque pueden editarse en el directorio /etc

```
adrianradice:EJ1 👉 xxd imagen | head
00000000: 8950 4e47 0d0a 1a0a 0000 000d 4948 4452  .PNG........IHDR
00000010: 0000 0146 0000 0196 0806 0000 0004 e92a  ...F...........*
00000020: df00 0018 5069 4343 5049 4343 2050 726f  ....PiCCPICC Pro
00000030: 6669 6c65 0000 5885 9559 0934 55dd dbdf  file..X..Y.4U...
00000040: e7ce f79a ef35 cff3 4ce6 799e e779 26e1  .....5..L.y..y&.
00000050: 9aae 29ae 3194 4832 95a8 10a5 928c 954a  ..).1.H2.......J
00000060: a112 9106 a50c bd24 4924 43a5 5051 19f2  .......$I$C.PQ..
00000070: 1d43 fddf fffb aef5 7deb db6b ed73 7ef7  .C.....}..k.s~.
00000080: d9cf 7e86 bd9f 3d3c e702 c0b5 df37 3232  ..~...=<.....722
00000090: 0cc1 0840 7844 0cd5 c1d4 80df cddd 831f  ...@xD..........
adrianradice:EJ1 👉
```

PM format	.pm	56 49 45 57	VIEW
PNG format	.png	89 50 4e 47	.PNG
Postscript format	.[e]ps	25 21	%!

SHEBANG

Especifica el Path al interprete que va a procesar el código. Una vez añadido, el SO operativo va a dejar de ver al fichero cómo un archivo text/plain y comenzará a catalogarlo de acuerdo a esta información.

Para lograr esto, sólo hace falta agregar en la primera línea de cualquier archivo de texto plano la combinación de caracteres #! Seguido del path del interprete

BASH SCRIPTING

Autor: Radice Adrian

SCRIPT

Conjunto de instrucciones a ejecutar

ENTONCES CREEMOS NUESTRO PRIMER SCRIPT

HELLO WORLD

```
#!/bin/bash
echo "hello world"
adrianradice:bash 👉 chmod u+x Ej1/source.sh
adrianradice:bash 👉 ./Ej1/source.sh
hello world
adrianradice:bash 👉
```

Notar que la primera linea de nuestro script es el Magic Number
Luego utilizamos el comando echo para imprimir por pantalla

COMENTARIOS

El analizador léxico del interprete al encontrarse con un # analizara el próximo carácter si no se trata de un ! Considera todo lo que continua como un comentario hasta el salto de linea.
RESUMEN: En bash solo tenemos comentarios de una linea, y inician con el símbolo #

```
#!/bin/bash
#Intro a BASH
echo "hello world" #imprimir por pantalla
#FIN DE MI PRIMER SCRIPT
```

VARIABLES

Cómo se menciono atras, bash es un lenguaje dinámico, lo que implica por ejemplo que las variables son no tipadas y no requieren ser declaradas previamente.

```
#!/bin/bash
#Declaración de variable
v1=1.3 #FLOAT
v2=3 #INT
v1="hola" #STRING
v2=$v1 #STRING
echo $v1
```

Para acceder al contenido de una variable debemos anteponer al nombre de la misma el signo $, ejemplo $V1 es el contenido de V1

VARIABLES DE ENTORNO

Variable	Descripción
DISPLAY	Donde aparecen la salidas de X-Windows.
HOME	Directorio personal.
HOSTNAME	Nombre de la máquina.
MAIL	Archivo de correo.
PATH	Lista de directorios donde buscar los programas.
PS1	Prompt.
SHELL	Intérprete de comandos por defecto.
TERM	Tipo de terminal.
USER	Nombre del usuario.

Env => Lista el contenido de todas las variables de entorno
Export <VAR>=<nuevo val> => edita una variable de entorno

VARIABLES INTERNAS

$0 nombre del script
$1, $2, …, ${10},… Parametros enviados
$@ array de parametros
$* cadena de parámetros
$$ PID proceso actual
$! PID ultimo proceso hijo ejecutado en segundo plano
$? Salida del ultimo comando ejecutado

EJEMPLO

```bash
#!/bin/bash
echo "Cantidad de parametros recibidos: " $#
echo "Parametros recibidos: " $@
echo "El valor del parámetro 1 es: " $1
echo "El valor del parámetro 2 es: " $2
echo "Mi PID es: " $$
mkdir test
rm -r test
echo "Salida de rm: "$?
rm -r test
echo "Salida de rm: "$?
sleep 1 &
echo "El pide de mi hijo es: " $!
```

Cual sería la salida de ejecutar el script como
./ejemplo “hola mundo” Sisop Intro a bash

Analicemos

```
adrianradice:EJ1 👉 ./source.sh "hola mundo" Sisop Intro a bash
Cantidad de parametros recibidos:  5
Parametros recibidos:  hola mundo Sisop Intro a bash
El valor del parámetro 1 es:  hola mundo
El valor del parámetro 2 es:  Sisop
Mi PID es:  6871
Salida de rm: 0
rm: test: No such file or directory
Salida de rm: 1
El pide de mi hijo es:  6875
adrianradice:EJ1 👉
```

Son 5 parámetros porque el delimitado de campos IFS por defecto es el “ “.
Notar que bash separa a Sisop Intro a Bash en 4 parámetros.
Pensar que pasa si ejecuto mkdir $1 vs mkdir “$1”

Analicemos

```bash
#!/bin/bash
echo "Cantidad de parametros recibidos: " $#
echo "Parametros recibidos: " $@
echo "El valor del parámetro 1 es: " $1
echo "El valor del parámetro 2 es: " $2
echo "Mi PID es: " $$
mkdir test
rm -r test
echo "Salida de rm: "$?
rm -r test
echo "Salida de rm: "$?
sleep 1 &
echo "El pide de mi hijo es: " $!
```

```
adrianradice:EJ1 👉 ./source.sh "hola mundo" Sisop Intro a bash
Cantidad de parametros recibidos:  5
Parametros recibidos:  hola mundo Sisop Intro a bash
El valor del parámetro 1 es:  hola mundo
El valor del parámetro 2 es:  Sisop
Mi PID es:  6871
Salida de rm: 0
rm: test: No such file or directory
Salida de rm: 1
El pide de mi hijo es:  6875
adrianradice:EJ1 👉
```

Observar que cuando un comando tiene una ejecución exitosa su salida es 0 de lo contrario es <>0.
Esto a se debe que para indicar que algo salió bien alcanza un numero, en Cambio los errores pueden ser muchos y deberíamos poder diferenciarlos, entonces tenemos distintos valores asignado a cada error.

FLUJOS

```bash
#!/bin/bash
echo "Cantidad de parametros recibidos: " $#
echo "Parametros recibidos: " $@
echo "El valor del parámetro 1 es: " $1
echo "El valor del parámetro 2 es: " $2
echo "Mi PID es: " $$
mkdir test
rm -r test
echo "Salida de rm: "$?
rm -r test
echo "Salida de rm: "$?
sleep 1 &
echo "El pide de mi hijo es: " $!
```

```
adrianradice:EJ1 👉 ./source.sh "hola mundo" Sisop Intro a bash
Cantidad de parametros recibidos:  5
Parametros recibidos:  hola mundo Sisop Intro a bash
El valor del parámetro 1 es:  hola mundo
El valor del parámetro 2 es:  Sisop
Mi PID es:  6871
Salida de rm: 0
rm: test: No such file or directory
Salida de rm: 1
El pide de mi hijo es:  6875
adrianradice:EJ1 👉
```

Cómo editaríamos el script para ignorar el mensaje de error de rm?

```
10  rm -r test 2>/dev/null
```

```
adrianradice:EJ1 👉 ./source.sh "hola mundo" Sisop Intro a bash
Cantidad de parametros recibidos:  5
Parametros recibidos:  hola mundo Sisop Intro a bash
El valor del parámetro 1 es:  hola mundo
El valor del parámetro 2 es:  Sisop
Mi PID es:  6961
Salida de rm: 0
Salida de rm: 1
El pide de mi hijo es:  6965
```

El 2 indica que utilizaremos la salida STDERR
El Símbolo > indica que el contenido de 2 se vuelque en /dev/null
/dev/null es la alfombra que esconde la basura en Linux

REDIRECCIONANDO FLUJOS

<contenido> > <destino> Crear un archivo nuevo con el contenido. Si no se indica el descriptor toma por defecto al 1

<contenido> >> <destino> Crear un archivo nuevo con el contenido o agregar el contenido al existente. Si no se indica el descriptor toma por defecto al 1

| (PIPE) alimenta el STDIN de un comando con el STDOUT de otro

EJEMPLO

```
adrianradice:EJ1 👉 ls
source.sh
adrianradice:EJ1 👉 date > "salida date"
adrianradice:EJ1 👉 ls
salida date     source.sh
adrianradice:EJ1 👉 cat "salida date"
Tue Apr  2 09:51:14 -03 2019
adrianradice:EJ1 👉 date > "salida date"
adrianradice:EJ1 👉 ls
salida date     source.sh
adrianradice:EJ1 👉 cat "salida date"
Tue Apr  2 09:51:42 -03 2019
adrianradice:EJ1 👉 date >> "salida date"
adrianradice:EJ1 👉 cat "salida date"
Tue Apr  2 09:51:42 -03 2019
Tue Apr  2 09:52:00 -03 2019
adrianradice:EJ1 👉 date | cut -d " " -f 1,2,4
Tue Apr 2
```

TOMA LA SALIDA DE DATE Y LA ESCRIBE EN UN ARCHIVO

PISAMOS EL ARCHIVO ANTERIOR CON LA NUEVA SALIDA DE DATE

AGREGAMOS AL ARCHIVO ANTERIOR UNA NUEVA FECHA

TOMAMOS LA SALIDA DE DATE Y MEDIANTE EL COMANDO CUT CONSERVAMOS LAS PARTES QUE NOS INTERESA

FUNCIONES

LAS FUNCIONES SE DEFINEN ANTES DE LLAMARSE

LOS PARAMETROS DE LAS FUNCIONES NO SE DEFINEN Y SE TRATAN IGUAL QUE LOS PARAMETROS DE LOS SCRIPT

```
nombreFuncion() {
        #Contenido de la funcion
}
```

OJO! EL SCOPE DE LAS VARIABLES ES GLOBAL EN BASH

ANALICEMOS COMO AFECTA EL SCOPE EN EL SIGUIENTE SCRIPT

```
adrianradice:EJ1 👉 cat s.sh
#!/bin/bash

test(){
        echo $nombre $1
        nombre="oscar"
}

nombre="adrian"

echo "\$1 $1"

test radice

echo $nombre
```

```
adrianradice:EJ1 👉 ./s.sh hola
```

```
$1 hola
adrian radice
oscar
adrianradice:EJ1 👉
```

SENTENCIAS DE CONTROL

IF

- Estructura:

```
if <condition> ; then
    <commands>
elif <condition>; then
    <commands>
else
    <commands>
fi
```

- ¿Como funciona? Evalúa la salida de un comando.
- La salida de un comando en el mundo Linux es: 0 => True ; >0 => False

Les parecerá raro que True es 0. Él porque es porque como se indico es la salida de un comando lo que se está evaluando, y los errores son los que se enumeran. Recordemos un poco de C

```
#include<stdio.h>
int main(){
    int i;
    printf("Ingrese un numero positivo para continuar: ");
    scanf("%d",&i);
    if(i<0)
        return 1; //EXIT_FAILURE;
    ....
    return 0; //EXIT_SUCCESS;
}
```

IF

- Como escribimos las condiciones Forma larga:

```
if test <comparacion>; then
fi
```

test si la condición es verdadera retornara 0 y sino >0, es el comando evaluado

Una manera más corta, es remplazar test por **[]**

```
if test $a -ge 6; then
fi
# es equivalente a
if [ $a -ge 6 ]; then
fi
```

IF

Operador	Nombre	Ejemplo	Equivalente C	Uso
-eq	**Eq**uality	2 -eq 2	==	Compara si dos objetos son iguales
-ne	**N**ot **E**qual	4 -ne 2	!=	Compara si dos objetos son distintos
-lt	**L**ess **T**han	1 -lt 2	<	Compara si un objeto es menor a otro
-le	**L**ess Than or **E**qual To	1 -le 5	<=	Compara si un objeto es menor igual a otro
-gt	**G**reater **T**han	2 -gt 1	>	Compara si un objeto es mayor a otro
-ge	**G**reater Than or **E**qual To	2 -ge 1	>=	Compara si un objeto es mayor igual a otro

Los operadores and y or son -a y -o.

```
if [ $a -ge 6  -a $a -le 9 ]; then
fi
```

IF

Bash admite la sintaxis de doble paréntesis para comparar números enteros únicamente, usando los operadores == >= && || ... en lugar de los -eq ... Analogía

- (()) => let
- [] => test

```
if test $a -ge 6; then
fi
# es equivalente a
if [ $a -ge 6 ]; then
fi
# es equivalente a
if (( $a == 6 )); then
fi
```

Negar una condición implica el operador !

```
if [ ! $a -ge 6 ]; then
fi
```

IF

Otra sintaxis es la de doble corchete y la misma habilita el uso de regex. Algunas consideraciones para esta última sintaxis

```
if [[ "$var" == *cadena* ]]; then
fi
# NO ES EQUIVALENTE A, EL PRIMER CASO EVALUA LA QUE LA VARIABLE CONTENGA LA CADENA, MIENTRAS QUE EL SEGUNDO QUE S
if [ "$var" -eq *cadena* ]; then
fi
```

Regex y operador ~=

- `.` Significa cualquier caracter.
- `^` Indica el principio de una línea.
- `$` Indica el final de una línea.
- `*` Indica cero o más repeticiones del caracter anterior.
- `+` Indica una o más repeticiones del caracter anterior.
- `\<` Indica el comienzo de una palabra.
- `\>` Indica el final de una palabra.
- `\` Caracter de escape. Da significado literal a un metacaracter.
- `[ ]` Uno cualquiera de los caracteres entre los corchetes. Ej: [A-Z] (desde A hasta Z).
- `[^ ]` Cualquier caracter distinto de los que figuran entre corchetes: Ej: [^A-Z].
- `{ }` Nos permiten indicar el número de repeticiones del patrón anterior que deben darse.
- `|` Nos permite indicar caracteres alternativos: Ej: (^|[?&])
- `( )` Nos permiten agrupar patrones. Ej: ([0-9A-F]+:)+

```
if [[ $var ~= ^[0-9]+$ ]]; then
   echo numero
fi
```

IF

Operador	Uso
-d FILE	True si existe y es un directorio
-e FILE	True si existe
-f FILE	True si existe y es un archivo regular
-h FILE	True si existe y es un SW Link
-p FILE	True si existe y es un pipe
-r FILE	True si tiene permiso de lectura
-w FILE	True si tiene permiso de escritura
-x FILE	True si tiene permiso de ejecución
-z CADENA	Cadena vacía
-n CADENA	Cadena no vacía
-s FILE	True si existe y su longitud es mayor a 0

CASE

```
case $variable in
     pattern-1)
          commands
          ;;
     pattern-2)
          commands
          ;;
     pattern-3|pattern-4|pattern-5)
          commands
          ;;
     pattern-N)
          commands
          ;;
     *)
          commands
          ;;
esac
```

SIMILITUD CON C

- SWITCH
- CASE
- BREAK
- CASE ANIDADOS
- DEFAULT

EJEMPLO

Ejercicio
Realizar un script que permita crear o eliminar un directorio
El mismo deberá ofrecer una ayuda de uso

PRIMERO ACORDAMOS COMO IDENTIFICAR LAS OPERACIONES

-c Crear directorio
-r Eliminar directorio
-h recibir ayuda

./ej -c “test sisop”

./ej -r “test sisop”

./ej -h

DEFINAMOS LA SECUENCIA DE NUESTRO SCRIPT

```
cantPar=$#
validarParametros $1
case $1 in
    -r)
        ruta=$2
        eliminar
    ;;
    -c)
        ruta=$2
        crear
    ;;
    -h|--help|-?)
        ayuda
    ;;
    *)
        echo "parametro invalido"
    ;;
esac
```

DEFINAMOS LA VALIDACION DE PARAMETROS

```
#!/bin/bash

function validarParametros(){
    if [ $cantPar -gt 2 ] # Supera la cantidad maxima de parametros
    then
        echo "Demasiados parametros"
        exit
    fi
    if [ $cantPar -eq 0 ] # no alcanza la cantidad minima de parametros
    then
        echo "revise los parametros necesarios"
        exit
    fi

    if [ $cantPar -eq 1 ]
    then
        if ! [ $1 = -h -o  $1 = --help -o $1 = ? ]
        then
            echo "Parametros invalidos"
            exit
        fi
    fi

}
```

FINALMENTE

```bash
#!/bin/bash

function validarParametros(){
    if [ $cantPar -gt 2 ] # Supera la cantidad maxima de parametros
    then
        echo "Demasiados parametros"
        exit
    fi
    if [ $cantPar -eq 0 ] # no alcanza la cantidad minima de parametros
    then
        echo "revise los parametros necesarios"
        exit
    fi

    if [ $cantPar -eq 1 ]
    then
        if ! [ $1 = -h -o  $1 = --help -o $1 = ? ]
        then
            echo "Parametros invalidos"
            exit
        fi
    fi

}

function ayuda(){
    echo "Proposito:"
    echo "  Eliminar o crear un directorio"
    echo "Parametros"
    echo "  -r eliminar directorio"
    echo "  -c crear directorio"
    echo "  -h|--help|-? ayuda"
}

crear(){
    mkdir "$ruta"
}

eliminar(){
    if `rm -rf "$ruta" 2>/dev/null`
    then
    echo ok
    else
    echo "error"
    fi
}

cantPar=$#
validarParametros $1
case $1 in
    -r)
        ruta=$2
        eliminar
    ;;
    -c)
        ruta=$2
        crear
    ;;
    -h|--help|-?)
        ayuda
    ;;
    *)
        echo "parametro invalido"
    ;;
esac
```

NOTA: EJEMPLO DE AYUDA PARA SCRIPT!!!!!!

Autor: Radice Adrian

CICLOS

while:

- Ejecuta una seria de comandos mientras que una determinada condición sea cumpla.
- Sintax

```
while [ <condicion> ]; do
 # codigo
done
```

until:

- Ejecuta una serie de comandos hasta que una determinada condición se cumpla.
- Sintax

```
until [ <condicion> ]; do
 # codigo
done
```

CICLOS

for:

- Ejecuta una serie de comandos un numero determinado de veces.
- Sintax: tenemos varias formas:
- Equivalente a c:

```
for (( ; ; )) ## SI NO COMPLETAMOS ES UN LOOP INFINITO
do
   #codigo
done
```

- Iterando sobre resultado de un comando
-

```
for OUTPUT in $(Linux-Or-Unix-Command)
do
   #codigo
done
```

CICLOS

- Iterando sobre listas
-

```
# Lista numerica explicita
for OUTPUT in 1 2 5 6 78 9
do
   #codigo
done
# Lista numerica generada explicita
for OUTPUT  in {0..10..2} #{comienzo..fin..incremento} el incremento por defecto es 1 y puede no especificarse
do
   #codigo
done
# Iterando sobre un script
IFSVIEJO=$IFS
IFS='-' ## DELIMITADOR DE CAMPOS QUE POR DEFAULT ES EL ESPACIO
for OUTPUT in  "HOLA-MUNDO"
do
   #codigo
done
IFS=$IFSVIEJO
```

PLUS EJEMPLO CONSERVACION DEL IFS

ARREGLOS

Características

- No tienen un límite fijo de tamaño (Solo el espacio disponible en la memoria lo limita).
- Se indexan a partir del 0.
- No se almacenan contiguamente en memoria Arreglos dispersos.
- Son Unidimensionales.

Tipos

- Indexados (a)
- Asociativos (A)

Indexado -> Se accede por posición
Asociativo -> Se accede por clave

ARREGLOS OPERACIONES

- Crear un array vacio, en forma indirecta

```
vec=()
```

- Crear un array inicializado, en forma indirecta

```
vec=('chau',123,'UNLAM')
```

- Crear un array inicializado, en forma indirecta, indicando los indices

```
vec=([0]='chau' [3]=123 [1]='UNLAM')
```

- Crear un array vacío, en forma explicita

```
declare -a vecIndexado
declare -A vecAsociativo
```

- Crear un array en forma explícita y inicializado

```
declare -a vec=('hola','tu',123.12)
```

Nota: Se crea un vector de 3 elementos [0]=>'hola';[1]=>'tu';[2]=>123.12

- Crear un array a partir de otro

```
vecCopia=("${vecOriginal[*]}")
vecCopiaConcat=("${vec1[*]}" "${vec2[*]}" )
vecCopiaEditado=("${vecOriginal[*]/valorAremplzar/nuevoValor}")
```

OPERACIONES ARREGLOS

- Convertir una variable en array

```
var=2
var[1]=21
```

Nota: Cualquier variable se puede considerar como un arreglo de un elemento

- Agregar un elemento a un array

```
vec+=(<lista de nuevos elementos>)
```

OPERACIONES ARREGLOS

Limpiar arreglo

- Liberar espacio array

```
unset -v array
```

- Liberar un elemento

```
unset -v array[1]
```

OPERACIONES ARREGLOS

Acceder a elementos de un arreglo

vec = (	a	, b	, c	, d	)
Index +	0	1	2	3	
Index -	-4	-3	-2	-1	

```
# Indices positivos
for ((i=0;i<4;i++)); do
    echo ${array[$i]}
done
# Indices negativos
for ((i=-1;i>=-4;i--)); do
    echo ${array[$i]}
done
# Mostrar todos los elementos
echo ${array[*]}
echo ${array[@]}

#Acceder a un rango de elementos
echo ${array[@]:1:2}
```

OPERACIONES ARREGLOS

- Obtener longitud del array

```
echo ${#array[@]}
```

- Obtener indices del array

```
echo ${!array[*]}
```

- Decir cuál de las dos siguientes formas es la correcta para incrementar en uno los elementos de un array*

```
for item in ${array[$i]}; di
    (( item = $item + 1 ))
done

for ((i=-1;i>=-4;i--)); do
    (( array[$i] = ${array[$i]} +1 ))
done
```

NOTA @ expande cada elemento como una cadena
* expande el array como una cadena

```
adrianradice@MacBook-Air-de-Adrian ~ % IFS=:
adrianradice@MacBook-Air-de-Adrian ~ % echo "${array[@]}"
a b c 1 2
adrianradice@MacBook-Air-de-Adrian ~ % echo "${array[*]}"
a:b:c:1:2
adrianradice@MacBook-Air-de-Adrian ~ %
```

Autor: Radice Adrian

Señales

- Enviar una señal

```
kill [señal] [PID]
```

- Consultar listado de señales

```
kill -l
```

- Consultar información de una señal en particular

```
man [numeroSeñal] signal
```

- Capturar una señal

```
trap [funcion] [listado señales separados por espacio]
```

- Restablecer comportamiento al capturar una señal

```
trap [señal a restablecer]
```

NO TODAS LAS SEÑALES PUEDEN SER CAPTURADAS

DEMONIO

QUE ES UN DEMONIO?

Demonios

Un demonio podría entenderse como un proceso ejecutándose en segundo plano. Para ejecutar un script en segundo plano, debemos utilizar el operador &

```
./miScript.sh &
```

Un demonio básicamente podríamos entenderlo como un loop infinito ejecutando un conjunto de tareas.

Pensemos la estructura básica para hacer un demonio:

```
#!/bin/bash
echo PID: $$
while true
do
  # Grupo de tareas que queremos que se ejecuten.
done
```

QUÉ PROBLEMA VEN?

Ahora bien, observemos que, si dejamos el demonio como esta, sería poco eficiente porque estaría consumiendo casi todo el tiempo del cpu, para ello tenemos dos caminos a optar, la elección del mismo dependerá si el demonio responderá rápidamente o no a un estímulo externo.

- Si el tiempo de respuesta no es crítico podemos usar la siguiente estructura:

```
#!/bin/bash
echo PID: $$
while true
do
  # Grupo de tareas que queremos que se ejecuten.
  sleep <algun tiempo>
done
```

- Si se requiere una respuesta inmediata podemos aprovechar el comando wait que es asincrónico,

```
#!/bin/bash
echo PID: $$
while true
do
  sleep <algun tiempo> & #NOTAR QUE ESTOY CREANDO UN SUBPROCESO
  wait $! #Espero asincronicamente por el subproceso
done
```

Finalmente, a nuestro demonio nos faltaría poder, obtener una finalización controlada, para ello podemos emplear lo visto en señales

demonio.sh

```bash
#!/bin/bash

function signal_SIGUSR1
{
    echo "RX=>USR1."
}

function signal_SIGUSR2
{
    echo "RX=>USR2. SIGINT HABILITADA"
    trap SIGINT
}

echo "PID: $$"

# Conecta las señales con las funciones.
trap signal_SIGUSR1 SIGUSR1
trap signal_SIGUSR2 SIGUSR2

# Ignora SIGINT ctrl-c
trap '' SIGINT

read a
```

COMPRIMIR UN ARCHIVO

```
Archivos .tar.gz:
Comprimir: tar -czvf empaquetado.tar.gz /carpeta/a/empaquetar/
Descomprimir: tar -xzvf archivo.tar.gz

Archivos .tar:
Empaquetar: tar -cvf paquete.tar /dir/a/comprimir/
Desempaquetar: tar -xvf paquete.tar

Archivos .gz:
Comprimir: gzip -9 index.php
Descomprimir: gzip -d index.php.gz

Archivos .zip:
Comprimir: zip archivo.zip carpeta
Descomprimir: unzip archivo.zip
```