

# Trabajo Práctico - Administración de Sistemas Linux con Vagrant

## Entrega Final

**Fecha límite:** 3 de julio de 2025 (inclusive)

⚠ **IMPORTANTE:** Los commits realizados después del 3 de julio de 2025 no serán tenidos en cuenta para la evaluación.

---

### ⚠ **IMPORTANTE - Usuarios de Mac M1/M2**

Si tienes un Mac con chip M1 o M2, debes usar QEMU en lugar de VirtualBox. Sigue estas instrucciones:

#### 1. Instalar QEMU:

```
bash  
  
brew install qemu
```

#### 2. Usar el Vagrantfile específico para ARM (se proporciona más abajo)

## Distribución de Calificaciones

- **30%** - Tareas Individuales
- **40%** - Tareas Grupales/Colaborativas
- **10%** - Claridad en Presentación y Documentación
- **20%** - Ejercicio Bonus (Compilación de Kernel)

## Introducción

Este trabajo práctico está diseñado para equipos de **3 integrantes** que trabajarán de forma colaborativa en un entorno Linux virtualizado. Si tu grupo tiene menos de 3 personas, **deberás asumir los roles y tareas de los perfiles faltantes**.

## Objetivos de Aprendizaje

- Virtualización con Vagrant
- Control de versiones colaborativo con Git (comandos básicos)
- Administración básica de sistemas Linux
- Gestión de permisos y usuarios
- Administración de discos y sistemas de archivos
- Contenedores con Docker y Docker Compose

### ⚠ **Nota sobre Scripts**

Este trabajo **NO requiere** conocimiento previo de scripting. Todos los comandos se proporcionan paso a paso.

## Configuración Inicial

### Vagrantfile para Intel/AMD (VirtualBox)

```
ruby

Vagrant.configure("2") do |config|
  config.vm.box = "ubuntu/jammy64"
  config.vm.hostname = "linux-practice"

  config.vm.provider "virtualbox" do |vb|
    vb.memory = "2048"
    vb.cpus = 2
    # Agregar disco adicional de 1GB para ejercicios
    unless File.exist?('./disk1.vdi')
      vb.customize ['createhd', '--filename', './disk1.vdi', '--size', 1024]
    end
    vb.customize ['storageattach', :id, '--storagectl', 'SCSI', '--port', 2, '--device', 0, '--
  end

  config.vm.network "private_network", ip: "192.168.56.10"
  config.vm.network "forwarded_port", guest: 80, host: 8080
  config.vm.network "forwarded_port", guest: 53, host: 5353, protocol: "udp"

  config.vm.provision "shell", inline: <<-SHELL
    apt-get update
    apt-get install -y neofetch git docker.io docker-compose
    usermod -aG docker vagrant
    systemctl enable docker
    systemctl start docker
  SHELL
end
```

### Vagrantfile para Mac M1/M2 (QEMU)

ruby

```
Vagrant.configure("2") do |config|
  config.vm.box = "perk/ubuntu-2204-arm64"
  config.vm.hostname = "linux-practice"

  config.vm.provider "qemu" do |qe|
    qe.memory = "2048"
    qe.cpus = 2
    qe.arch = "aarch64"
    qe.machine = "virt,accel=hvf,highmem=off"
    qe.cpu = "cortex-a72"
    qe.net_device = "virtio-net-pci"
  end

  config.vm.network "private_network", ip: "192.168.56.10"
  config.vm.network "forwarded_port", guest: 80, host: 8080
  config.vm.network "forwarded_port", guest: 53, host: 5353, protocol: "udp"

  config.vm.provision "shell", inline: <<-SHELL
    apt-get update
    apt-get install -y neofetch git docker.io docker-compose
    usermod -aG docker vagrant
    systemctl enable docker
    systemctl start docker
    # Crear archivo que simule un disco adicional
    dd if=/dev/zero of=/tmp/disk1.img bs=1M count=1024
    losetup /dev/loop10 /tmp/disk1.img
  SHELL
end
```

## Distribución de Roles

### Alumno A - "Administrador"

- Responsable de la configuración inicial del sistema
- Gestión de usuarios y permisos
- Administración de discos

### Alumno B - "Desarrollador"

- Configuración de repositorio Git
- Gestión de contenedores Docker
- Automatización de tareas

## Alumno C - "Operador"

- Desarrollo de comandos
- Gestión de archivos y directorios
- Testing y validación

## Ejercicios

### 1. Configuración Inicial y Repositorio (Colaborativo)

#### Pasos:

1. **Alumno B** crea un repositorio público en GitHub llamado `practica-linux-[apellidos-equipo]`
2. Cada alumno clona el repositorio en su máquina virtual
3. Configurar Git con sus datos personales en cada VM
4. Crear la estructura inicial:

```
proyecto/  
├── informacion/  
├── permisos/  
├── discos/  
├── archivos/  
└── contenedores/
```

**Entregable:** Screenshot del repositorio creado y estructura de carpetas

### 2. Neofetch Colaborativo (Colaborativo)

**Objetivo:** Cada alumno debe ejecutar `neofetch` en su VM y agregar la salida al mismo archivo sin sobrescribir el contenido anterior.

#### Pasos:

1. Cada alumno ejecuta: `neofetch > temp_neofetch_[nombre].txt`
2. **Alumno A** inicia creando `neofetch/system_info.txt` con su neofetch
3. **Alumno B** hace pull, agrega su neofetch al final del archivo usando `cat`
4. **Alumno C** repite el proceso
5. El archivo final debe mostrar los 3 neofetch claramente separados

**Script de ayuda:**

```
bash
```

```
#!/bin/bash
echo "===== NEOFETCH DE [NOMBRE] =====" >> neofetch/system_info.txt
neofetch >> neofetch/system_info.txt
echo "" >> neofetch/system_info.txt
```

**Entregable:** Archivo `system_info.txt` con los 3 neofetch y historial de commits

### 3. Gestión de Permisos (Individual + Colaborativo)

#### Parte Individual (cada alumno):

1. Crear un directorio personal: `/home/vagrant/[apellido]_espacio/`
2. Crear un archivo `privado.txt` con permisos 600 (solo el dueño puede leer y escribir)
3. Crear un archivo `publico.txt` con permisos 644 (dueño lee/escribe, otros solo leen)
4. Crear usuarios locales para simular el trabajo colaborativo

#### Parte Colaborativa - Creación de usuarios locales:

Cada alumno debe crear usuarios locales en su máquina para simular el trabajo en equipo:

```
bash

# Cada alumno ejecuta estos comandos para crear usuarios de prueba:
sudo useradd -m estudiante1
sudo useradd -m estudiante2
sudo useradd -m estudiante3

# Verificar que se crearon correctamente:
cat /etc/passwd | grep estudiante
```

#### Trabajo con grupos:

1. **Alumno A** crea un grupo llamado `equipotrabajo`:

```
bash

sudo groupadd equipotrabajo
sudo usermod -a -G equipotrabajo estudiante1
sudo usermod -a -G equipotrabajo estudiante2
sudo usermod -a -G equipotrabajo estudiante3
sudo usermod -a -G equipotrabajo vagrant
```

2. Crear directorio colaborativo:

bash

```
sudo mkdir /tmp/colaborativo
sudo chgrp equipotrabajo /tmp/colaborativo
sudo chmod 770 /tmp/colaborativo
```

3. Cada alumno debe crear su archivo personal de verificación mostrando sus grupos:

bash

```
# Crear archivo con información personal de usuarios y grupos
echo "===== INFORMACIÓN DE USUARIOS Y GRUPOS - [APELLIDO] =====" > permisos/usuarios_[apellido]
echo "Usuario actual:" >> permisos/usuarios_[apellido].txt
whoami >> permisos/usuarios_[apellido].txt
echo "Grupos del usuario:" >> permisos/usuarios_[apellido].txt
groups >> permisos/usuarios_[apellido].txt
echo "ID del usuario:" >> permisos/usuarios_[apellido].txt
id >> permisos/usuarios_[apellido].txt
echo "Usuarios del sistema:" >> permisos/usuarios_[apellido].txt
cat /etc/passwd | grep estudiante >> permisos/usuarios_[apellido].txt
echo "" >> permisos/usuarios_[apellido].txt
```

¿Cómo hacer la verificación? La verificación consiste en ejecutar comandos y guardar su salida en un archivo de texto. El símbolo `>>` significa "agregar al final del archivo" y `>` significa "crear archivo nuevo o sobrescribir".

### Verificación de permisos:

bash

```
# Cada alumno ejecuta estos comandos para verificar Los permisos:
echo "===== VERIFICACIÓN DE PERMISOS - [APELLIDO] =====" >> permisos/verificacion_permisos.txt
echo "Archivos en mi directorio personal:" >> permisos/verificacion_permisos.txt
ls -la /home/vagrant/[apellido]_espacio/ >> permisos/verificacion_permisos.txt
echo "Archivos en directorio colaborativo:" >> permisos/verificacion_permisos.txt
ls -la /tmp/colaborativo/ >> permisos/verificacion_permisos.txt
echo "Información del grupo equipotrabajo:" >> permisos/verificacion_permisos.txt
getent group equipotrabajo >> permisos/verificacion_permisos.txt
echo "" >> permisos/verificacion_permisos.txt
```

**Entregable:** Archivos `usuarios_[apellido].txt` y `verificacion_permisos.txt` de cada alumno

## 4. Administración de Discos (Individual)

**Objetivo:** Cada alumno debe agregar, formatear y montar un disco adicional.

## Pasos para VirtualBox:

1. El disco ya está agregado por Vagrant (`/dev/sdc`)
2. Particionar el disco: `sudo fdisk /dev/sdc`
3. Formatear con ext4: `sudo mkfs.ext4 /dev/sdc1`
4. Crear punto de montaje: `sudo mkdir /mnt/almacenamiento_[apellido]`
5. Montar: `sudo mount /dev/sdc1 /mnt/almacenamiento_[apellido]`
6. Agregar al fstab para montaje automático

## Pasos para QEMU (Mac M1/M2):

1. Usar el loop device: `/dev/loop10`
2. Formatear: `sudo mkfs.ext4 /dev/loop10`
3. Continuar con el montaje igual que VirtualBox

**¿Cómo hacer la verificación de discos?** La verificación consiste en mostrar información sobre los discos montados y el espacio disponible.

## Verificación:

bash

*# Cada alumno ejecuta estos comandos para verificar los discos:*

```
echo "===== VERIFICACIÓN DE DISCOS - [APELLIDO] =====" >> discos/verificacion_discos.txt
echo "Espacio en disco:" >> discos/verificacion_discos.txt
df -h >> discos/verificacion_discos.txt
echo "Dispositivos de bloque:" >> discos/verificacion_discos.txt
lsblk >> discos/verificacion_discos.txt
echo "Configuración de montaje automático:" >> discos/verificacion_discos.txt
cat /etc/fstab | grep almacenamiento >> discos/verificacion_discos.txt
echo "" >> discos/verificacion_discos.txt
```

**Entregable:** Archivo `verificacion_discos.txt` con la verificación de cada alumno

## 5. Gestión de Archivos y Directorios (Individual)

**Objetivo:** Operaciones avanzadas con archivos y directorios.

### Tareas por alumno:

1. Crear estructura de directorios:

```
/mnt/almacenamiento_[apellido]/
├── proyectos/
│   ├── activos/
│   └── archivados/
├── respaldos/
└── temporal/
```

2. Crear 10 archivos de prueba en `temporal/`:

```
bash

cd /mnt/almacenamiento_[apellido]/temporal/
for i in {01..10}; do
    touch documento_${i}.txt
    echo "Contenido del documento $i" > documento_${i}.txt
done
```

3. Operaciones de copia y movimiento:

```
bash

# Copiar archivos 1-5 a proyectos/activos/
cp documento_01.txt documento_02.txt documento_03.txt documento_04.txt documento_05.txt ../proy

# Mover archivos 6-8 a proyectos/archivados/
mv documento_06.txt documento_07.txt documento_08.txt ../proyectos/archivados/

# Crear backup de archivos 9-10 en respaldos/
cp documento_09.txt documento_10.txt ../respaldos/

# Eliminar los archivos originales restantes de temporal/
rm documento_09.txt documento_10.txt
```

4. **¿Cómo hacer la verificación de archivos?** La verificación muestra todos los archivos creados y sus ubicaciones finales.

**Verificación:**



bash

# Cada alumno ejecuta estos comandos para verificar Los archivos:

```
echo "===== VERIFICACIÓN DE ARCHIVOS - [APELLIDO] =====" >> archivos/verificacion_archivos.txt
echo "Estructura de directorios creada:" >> archivos/verificacion_archivos.txt
find /mnt/almacenamiento_[apellido] -type d >> archivos/verificacion_archivos.txt
echo "Archivos en cada directorio:" >> archivos/verificacion_archivos.txt
find /mnt/almacenamiento_[apellido] -type f -exec ls -la {} \; >> archivos/verificacion_archivos.txt
echo "" >> archivos/verificacion_archivos.txt
```

**Entregable:** Archivo `verificacion_archivos.txt` con la verificación de cada alumno

## 6. Contenedores y Docker Compose (Colaborativo)

**Objetivo:** Levantar un servicio de Pi-hole usando Docker Compose.

**Asignación de tareas:**

- **Alumno A:** Investigar y crear el archivo `docker-compose.yml`
- **Alumno B:** Configurar variables de entorno y networking
- **Alumno C:** Testing y documentación

**archivo docker-compose.yml:**

```
yaml
version: '3.7'

services:
  pihole:
    container_name: pihole-practica
    image: pihole/pihole:latest
    ports:
      - "53:53/tcp"
      - "53:53/udp"
      - "80:80/tcp"
    environment:
      TZ: 'America/Argentina/Buenos_Aires'
      WEBPASSWORD: 'practica123'
      FTLCONF_LOCAL_IPV4: '192.168.56.10'
    volumes:
      - './configuracion-pihole:/etc/pihole'
      - './configuracion-dnsmasq.d:/etc/dnsmasq.d'
    restart: unless-stopped
    cap_add:
      - NET_ADMIN
```

## Tareas de verificación:

1. Ejecutar `docker-compose up -d`
2. Verificar que el servicio esté corriendo: `docker ps`
3. Acceder a la interfaz web (<http://192.168.56.10>)
4. Tomar screenshots de la interfaz
5. Verificar logs: `docker-compose logs`

**¿Cómo hacer la verificación de contenedores?** La verificación muestra que los contenedores están funcionando correctamente y responden a las peticiones.

## Verificación:

```
bash

# Cada alumno ejecuta estos comandos para verificar Docker:
echo "==== VERIFICACIÓN CONTENEDORES - [APELLIDO] =====>> contenedores/verificacion_contenec
echo "Contenedores en ejecución:">> contenedores/verificacion_contenedores.txt
docker ps >> contenedores/verificacion_contenedores.txt
echo "Estado de docker-compose:">> contenedores/verificacion_contenedores.txt
docker-compose ps >> contenedores/verificacion_contenedores.txt
echo "Prueba de conectividad al servicio:">> contenedores/verificacion_contenedores.txt
curl -I http://192.168.56.10 >> contenedores/verificacion_contenedores.txt
echo "">> contenedores/verificacion_contenedores.txt
```

## Entregable:

- Archivo `docker-compose.yml`
- Capturas de pantalla de Pi-hole funcionando
- Archivo `verificacion_contenedores.txt`

## Ejercicio Bonus (Opcional)

### Compilación de Kernel Personalizado

**Objetivo:** Uno de los integrantes del equipo compila un kernel personalizado y lo muestra con neofetch.

**Candidato:** El alumno con mejor rendimiento en ejercicios anteriores o por sorteo.

## Pasos:

1. Descargar código fuente del kernel: `wget https://cdn.kernel.org/pub/linux/kernel/v6.x/linux-6.1.tar.xz`
2. Configurar compilación: `make menuconfig` (cambiar solo la versión local)

3. Compilar: `make -j$(nproc)`

4. Instalar: `sudo make modules_install && sudo make install`

5. Actualizar bootloader: `sudo update-grub`

6. Reiniciar y verificar con `neofetch`

**Entregable:** Archivo `kernel_custom.txt` con neofetch mostrando el kernel compilado

## Criterios de Evaluación

### Trabajo Individual (30%)

- Completitud de ejercicios individuales
- Correcta ejecución de comandos
- Resolución de problemas básicos

### Trabajo Colaborativo (40%)

- Uso efectivo de Git (add, commit, push, pull)
- Resolución de conflictos básicos
- Comunicación y coordinación del equipo

### Presentación y Documentación (10%)


- Claridad de los archivos de verificación
- Screenshots y evidencias
- README del proyecto explicando el proceso

### Ejercicio Bonus (20%)

- Compilación exitosa del kernel personalizado
- Documentación del proceso
- Demostración con neofetch

## Entrega Final

**Fecha límite:** 3 de julio de 2025 (inclusive)

 **IMPORTANTE:** Los commits realizados después del 3 de julio de 2025 no serán tenidos en cuenta para la evaluación.

### Formato:

1. Repositorio Git con todos los archivos
2. README.md explicando el proceso y conclusiones

## Estructura del repositorio final:

```
proyecto/
├── README.md
├── Vagrantfile
├── informacion/
│   └── datos_sistema.txt
├── permisos/
│   ├── usuarios_[apellido].txt (uno por alumno)
│   └── verificacion_permisos.txt
├── discos/
│   └── verificacion_discos.txt
├── archivos/
│   └── verificacion_archivos.txt
├── contenedores/
│   ├── docker-compose.yml
│   ├── capturas/
│   └── verificacion_contenedores.txt
└── nucleo/ (bonus)
    └── informacion_nucleo.txt
```

## Recursos de Ayuda

- [Vagrant Documentation](#)
- [Git Collaboration Guide](#)
- [Docker Compose Reference](#)
- [Linux Commands Reference](#)

¡Buena suerte y que aprendan mucho! 🐧