Bahir Dar University

Faculty of Computing

Department of Software engineering

Software engineering

Project-work

NAME: Abel Amare

ID:1601012 SECTION:A

Installation Of Kwort Linux

Submitted to Wendimu Baye

Table of Contents

KWORT LINUX

What is an Operating System?

Introduction (Background and Motivation) of this project

An **Operating System (OS)** is the fundamental software layer that allows users and applications to interact with computer hardware. It manages system resources such as CPU, memory, storage, and peripheral devices, while providing essential services like process management, file systems, and security. The OS acts as a bridge between user-level programs and the physical machine.

The study of operating systems is crucial in **software engineering** because it enables us to understand how software runs, interacts with hardware, and responds to user or programmatic input. It also builds the foundation for advanced topics such as kernel programming, system calls, security, virtualization, and performance optimization.

This project focuses on **installing and configuring KWORT LINUX** — a lightweight, fast, and minimalist Linux distribution. The motivation behind choosing **KWORT LINUX** is its simplicity and transparency, which exposed me directly to core system components such as file systems, bootloaders, and run levels without abstraction. Unlike more automated operating systems (e.g., Ubuntu or Windows), **KWORT** requires manual configuration, making it ideal for an educational environment where **hands-on learning** is emphasized.

Moreover, performing the installation in a **virtual environment using VMware** added value by teaching me about **virtualization technologies**, which are essential in today's cloud-based and DevOps-driven world. Virtual machines provided a safe and flexible way to test operating systems without affecting the host system, allowing me to experiment, troubleshoot, and understand low-level processes without risk.

Through this project, I gain **practical skills** in:

- OS installation,
- Manual partitioning and file system management,
- **Config** file editing (e.g., `/etc/fstab`, `/etc/rc.conf`),
- Bootloader configuration,
- Networking and troubleshooting in Linux,
- And using virtualization as a powerful system programming tool.

In conclusion, this project bridges theoretical concepts of operating systems with real-world, practical implementation — building the foundation for future work in systems programming, cybersecurity, and infrastructure development.

What is KWORT Linux?

- ✓ **KWORT-LINUX:** is a minimalist, fast, and powerful Linux distribution based on CRUX (**N.B. CRUX is an operating system — specifically, a LINUX - DISTRIBUTION**. In the definition, "based on CRUX" simply means that KWORT Linux has inherited or incorporated some elements of CRUX.)

- ✓ **KWORT Linux** is specifically designed for users who want complete control over their system environment, offering a lightweight footprint and excellent performance. This project aims to understand the fundamentals of OS installation, virtual environments, and system functionality by setting up KWORT Linux in a virtualized environment. Originally released in **2004** by developer **David Cortarello**, Kwort has since evolved as a rolling-release distribution, receiving continuous updates rather than major version upgrades. It includes a custom package manager called **`kpkg`** and uses the **`runit` init system**, both of which contribute to its **speed, efficiency, and minimal resource usage**. Unlike mainstream **Linux distributions, Kwort** is intended for advanced users and learners who want to build their systems from the ground up, offering a valuable platform for deep system-level learning and hands-on experimentation .

    - **Having understood** the background of **KWORT Linux**, the following objectives were set to guide the practical and theoretical exploration of KWORT Linux in a virtual environment:

Objectives of this project

-To install and configure **KWORT** Linux using virtualization software (e.g., Virtual Box).

-To document the process, identify any issues, and explore solutions.

-To examine filesystem support and analyze advantages/disadvantages.

-To explore system-level operations using system calls like exec() .

-To develop technical problem-solving skills by encountering real-life system configuration challenges and resolving them using Linux command-line tools and configuration files.

-To evaluate the effectiveness of **KWORT** Linux as a platform for system programming and determine its suitability for future projects or academic research.

- **To achieve these objectives effectively,**

certain hardware and software components were essential.

The following section outlines the requirements necessary for successful implementation.

Requirements	
i. Hardware	**ii. Software**
CPU: x86_64-based processor (Intel/AMD)	Virtual Box
RAM: Minimum 1GB (2GB recommended)	**KWORT** Linux ISO
Storage: Minimum 10GB	Image editing/snipping tools for documentation

Virtual Machine: Oracle Virtual Box / VMware Workstation	Text editor (e.g., Vim, Nano, or graphical editor)

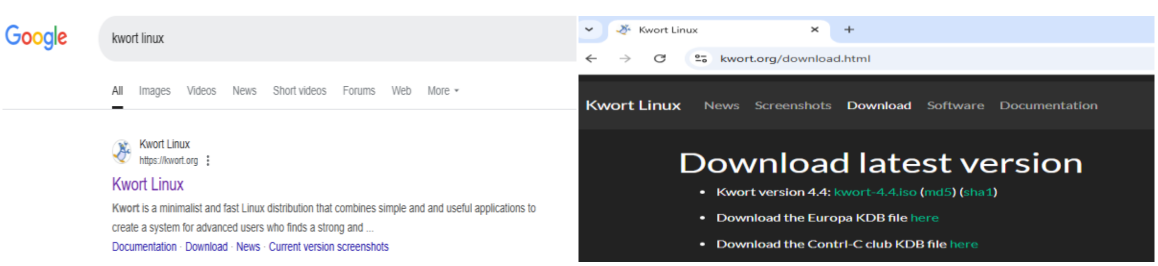
✓ With the necessary hardware and software requirements identified, the next step is to begin the actual installation process. This involves preparing the virtual environment in VMware, creating and formatting disk partitions, mounting the installation media, and configuring essential system files. The installation of KWORT Linux is largely manual, which gives students full control and understanding over each stage — from disk setup to bootloader configuration. The following steps outline this process in detail.

Installation Steps

This section outlines the complete step-by-step process of installing **KWORT Linux** in a **VMware Workstation Player** virtual machine. The goal is to gain practical experience in OS setup, partitioning, configuration, and bootloader installation.

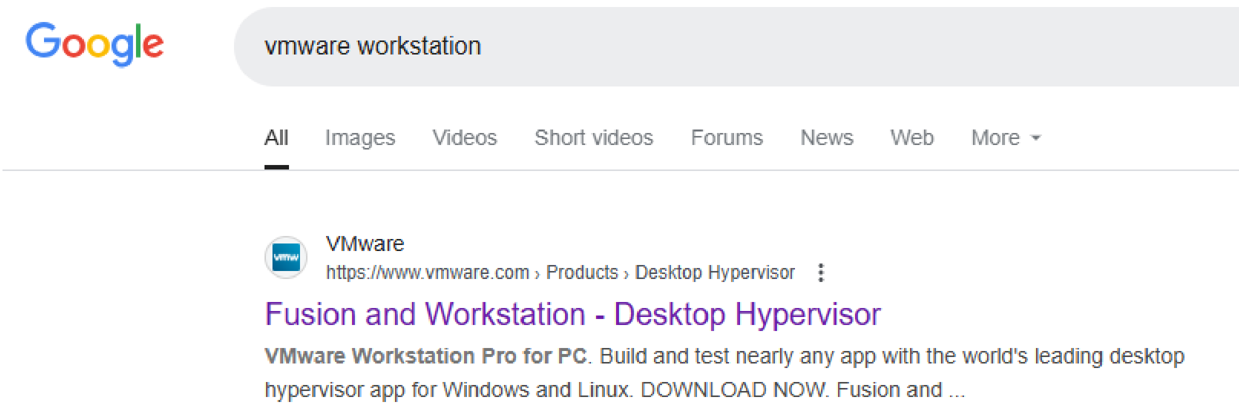
Step 1: • Visit the official **KWORT** website: https://kwort.org

- Click on the **"Download"** section.
- Select the latest ISO release (e.g., kwort-4.4.0.iso).
- Save it to a known location on your system (e.g., Downloads folder).

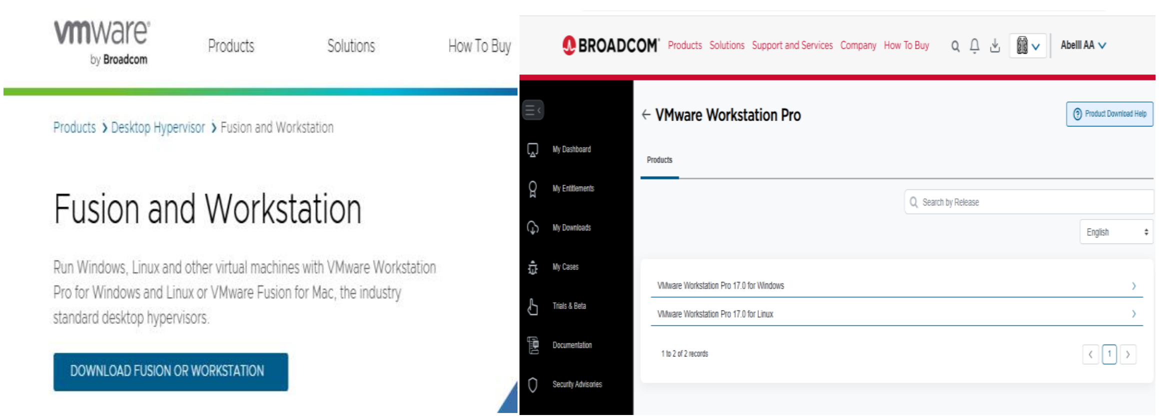


Step 2: Download and Install VMware Workstation Player

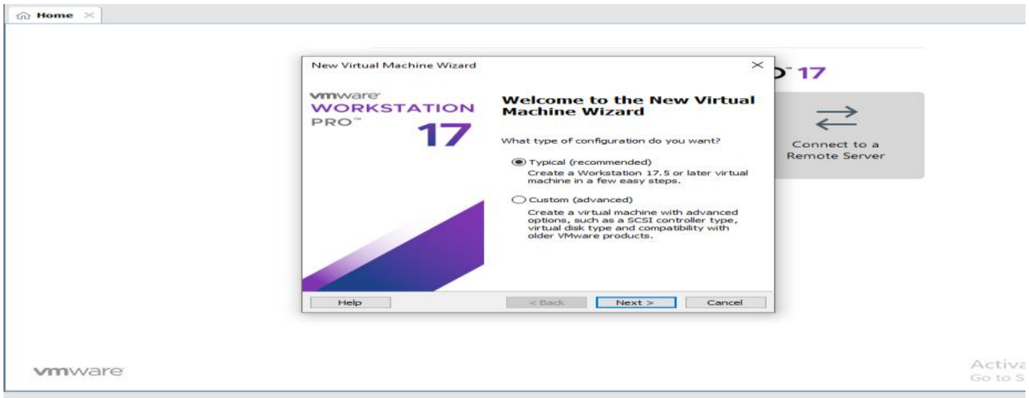
- Go to: https://www.vmware.com/go/getplayer
- Choose your OS (e.g., Windows or Linux). In my case it is Windows
- Download and install:

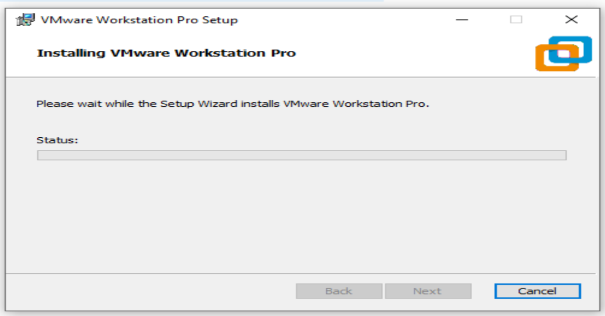


- ✓ After selecting the desired option, click the download button as shown below. This will redirect you to the website where the download is available then you will need to create an account using your Gmail address to access the site. Once you're signed in, you can proceed to download the software. **Note:** It is always better to choose the latest version before starting the download.

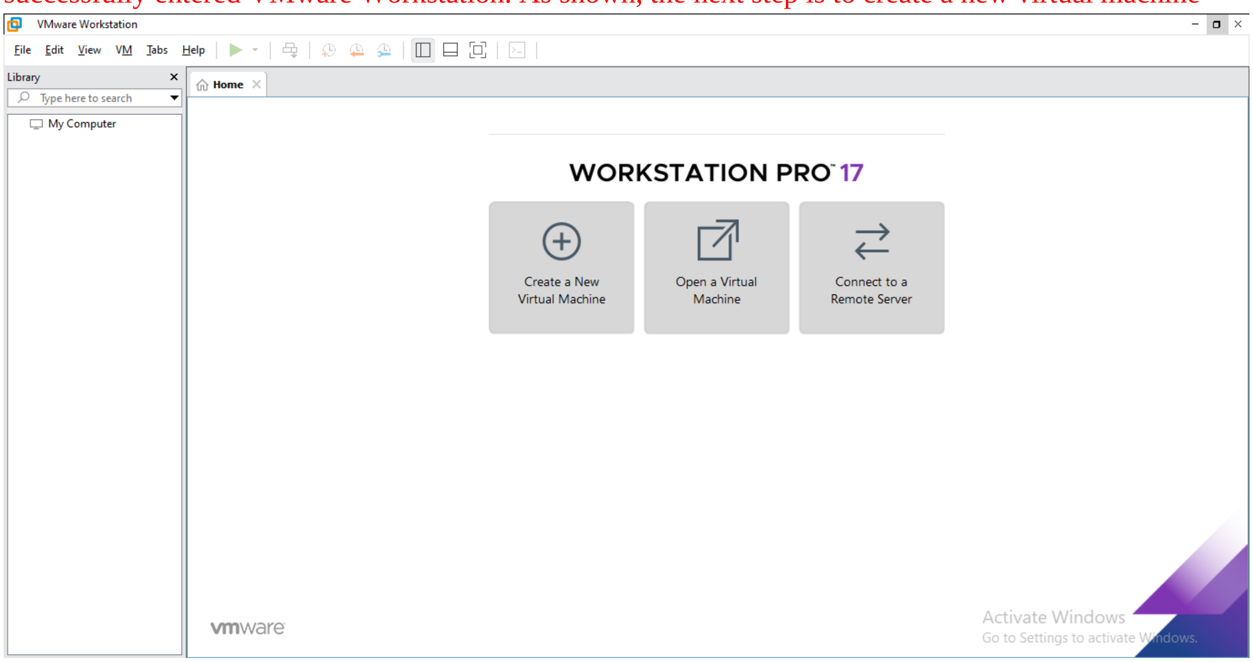


After completing the previous steps, navigate to the directory where the downloaded file is located and begin the installation. During the installation process, several setup options will appear. You can either configure them according to your preferences or simply click 'Next' to proceed with the default setup





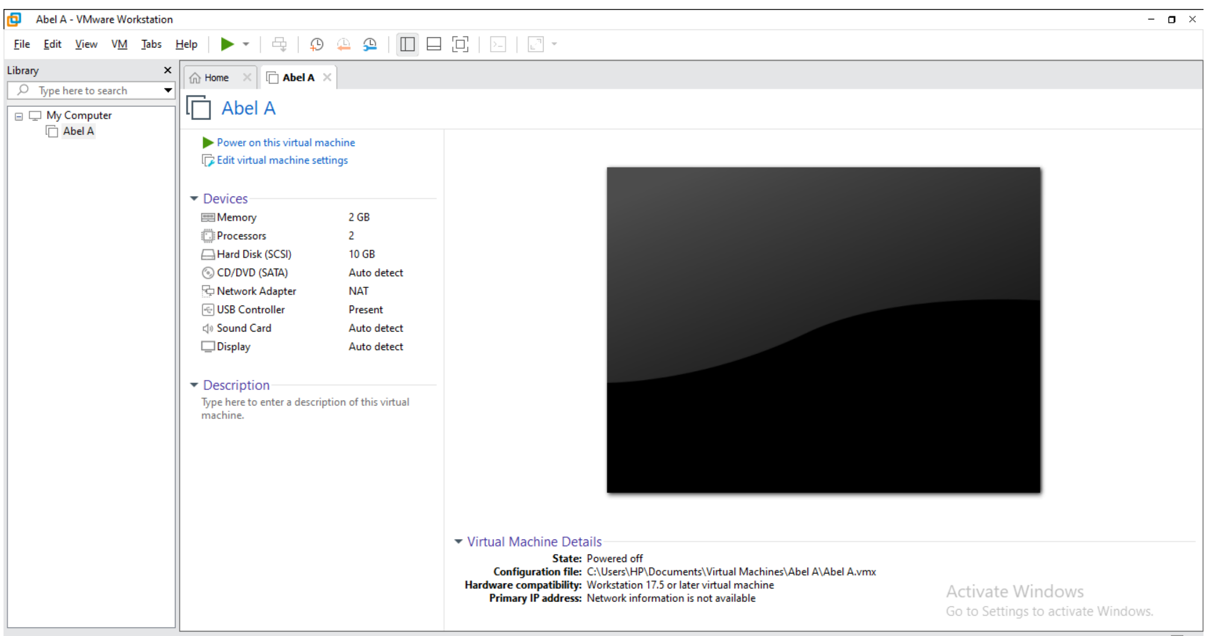
After completing these steps, you will see a screen like the one below, which means you have successfully entered VMware Workstation. As shown, the next step is to create a new virtual machine



Step 3: Create a New Virtual Machine

- Open **VMware Workstation Player**.
- Click on **“Create a New Virtual Machine”**.
- Select **“Installer disc image file (ISO)”** and choose the downloaded KWORT ISO.
- Set the OS type to **Linux** and version to **Other Linux 5.x or later kernel 64-bit**.
- Name your VM (e.g., KWORT-Linux in my case I chose Abel A) and choose a location to save it.
- Allocate:
    - Memory: **512MB or more**
    - Hard disk: **20GB (recommended for flexibility)**

It will be something like this

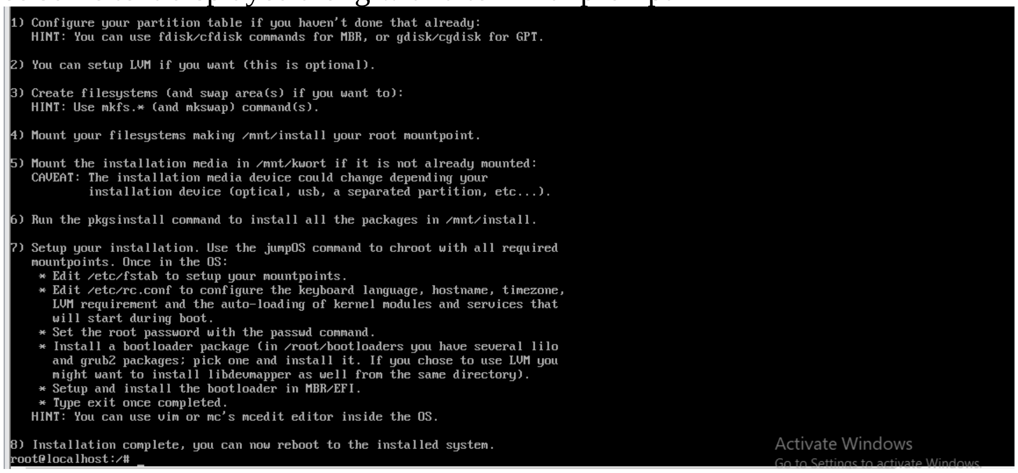


- Finish the setup and click **Power On** to start the VM

Step 4: Start KWORT Live Environment

- Once powered on, the VM will boot into the KWORT **live environment** automatically from the ISO.

□You will see some text displayed along with a terminal prompt



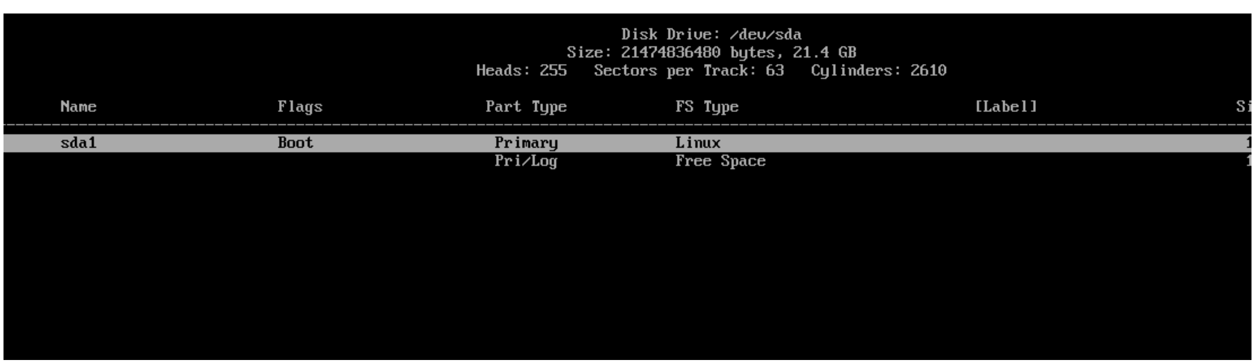
Step 5: Partition the Disk

Run: cfdisk /dev/sda

□ Create a **new primary partition**: by selecting Primary, then entering the desired size (e.g., 10G). After that, press Enter on the "Bootable option to mark the partition as bootable.

Next choose

□ Write then type yes



□ after you see boot on flags then and quit.

Step 6: Format the Partition

Run: mkfs.ext4 /dev/sda1

```
root@localhost:/# mkfs.ext4 /dev/sda1
mke2fs 1.42.9 (28-Dec-2013)
Filesystem label=
OS type: Linux
Block size=4096 (log=2)
Fragment size=4096 (log=2)
Stride=0 blocks, Stripe width=0 blocks
610800 inodes, 2441872 blocks
122093 blocks (5.00%) reserved for the super user
First data block=0
Maximum filesystem blocks=2503999488
75 block groups
32768 blocks per group, 32768 fragments per group
8144 inodes per group
Superblock backups stored on blocks:
        32768, 98304, 163840, 229376, 294912, 819200, 884736, 1605632

Allocating group tables: done
Writing inode tables: done
Creating journal (32768 blocks): done
Writing superblocks and filesystem accounting information: done

root@localhost:/# _
```

Step 7: Mount the Partition

Run: mount -t ext4 /dev/sda1 /mnt/install

```
root@localhost:/# mount /dev/sda1 /mnt/install_
```

Step 8: Run: pkgsinstall /mnt/kwort /mnt/install

```
oot@localhost:/# pkgsinstall
ackage /mnt/kwort/packages/filesystem#3.7#x86_64#1.tar.xz installed
ackage /mnt/kwort/packages/core/acl#2.3.1#x86_64#1.tar.xz installed
ackage /mnt/kwort/packages/core/attr#2.5.1#x86_64#1.tar.xz installed
ackage /mnt/kwort/packages/core/autoconf#2.71#x86_64#1.tar.xz installed
ackage /mnt/kwort/packages/core/automake#1.16.5#x86_64#1.tar.xz install
ackage /mnt/kwort/packages/core/bash#5.1.16#x86_64#1.tar.xz installed
ackage /mnt/kwort/packages/core/bc#1.07.1#x86_64#1.tar.xz installed
ackage /mnt/kwort/packages/core/binutils#2.38#x86_64#1.tar.xz installed
ackage /mnt/kwort/packages/core/bison#3.8.2#x86_64#1.tar.xz installed
ackage /mnt/kwort/packages/core/bzip2#1.0.8#x86_64#3.tar.xz installed
ackage /mnt/kwort/packages/core/ca-certificates#20220426#x86_64#1.tar.x
ackage /mnt/kwort/packages/core/cmake#3.23.1#x86_64#1.tar.xz installed
ackage /mnt/kwort/packages/core/coreutils#9.1#x86_64#1.tar.xz installed
ackage /mnt/kwort/packages/core/cpio#2.13#x86_64#2.tar.xz installed
ackage /mnt/kwort/packages/core/curl#7.83.1#x86_64#1.tar.xz installed
ackage /mnt/kwort/packages/core/dash#0.5.11.5#x86_64#1.tar.xz installed
ackage /mnt/kwort/packages/core/db#5.3.28#x86_64#2.tar.xz installed
ackage /mnt/kwort/packages/core/dcron#4.5#x86_64#3.tar.xz installed
ackage /mnt/kwort/packages/core/dhcpcd#9.4.1#x86_64#2.tar.xz installed
ackage /mnt/kwort/packages/core/diffutils#3.8#x86_64#1.tar.xz installed
ackage /mnt/kwort/packages/core/dumb_runtime_dir#1.0.4#x86_64#1.tar.xz
ackage /mnt/kwort/packages/core/e2fsprogs#1.46.5#x86_64#1.tar.xz instal
ackage /mnt/kwort/packages/core/ed#1.18#x86_64#1.tar.xz installed
ackage /mnt/kwort/packages/core/elfutils#0.187#x86_64#1.tar.xz installe
ackage /mnt/kwort/packages/core/eudev#3.2.11#x86_64#1.tar.xz installed
ackage /mnt/kwort/packages/core/expat#2.4.8#x86_64#1.tar.xz installed
ackage /mnt/kwort/packages/core/file#5.41#x86_64#1.tar.xz installed
ackage /mnt/kwort/packages/core/findutils#4.9.0#x86_64#1.tar.xz install
```

Step 9: Enter the Installed System

Run: **jumpOS /mnt/install**
Note:this will change root@localhost:/# to root@localhost:~#

```
root@localhost:/# jumpOS
root@localhost:~# _
```

Step 10: Inside the chroot: Configure the System

- **Edit /etc/fstab: vim /etc/fstab**

```
root@localhost:/#
root@localhost:/# jumpOS /mnt/install
root@localhost:~# vim /etc/fstab_
```

Then write, save and exit this: /dev/sda1 / ext4 defaults 1 1

```
#/dev/#EXT4FS_ROOT#    /          ext4      defaults
#/dev/#BTRFS_ROOT#     /          btrfs     defaults
#/dev/#XFS_ROOT#       /          xfs       defaults
#/dev/#F2FS_ROOT#      /          f2fs      defaults
#/dev/#SWAP#           swap       swap      defaults
#/dev/#EXT4FS_HOME#    /home      ext4      defaults
#/dev/#BTRFS_HOME#     /home      btrfs     defaults
#/dev/#XFS_HOME#       /home      xfs       defaults
#/dev/#F2FS_HOME#      /home      f2fs      defaults
#/dev/cdrom            /cdrom     iso9660   ro,user,noauto,unhide
#/dev/dvd              /dvd       udf       ro,user,noauto,unhide
#/dev/floppy/0         /floppy    vfat      user,noauto,unhide
tmp                   /tmp       tmpfs     defaults
usb                   /proc/bus/usb usbfs defaults

# the following entries are required for proper system operation
devpts                 /dev/pts   devpts    noexec,nosuid,gid=tty,mo
shm                    /dev/shm   tmpfs     defaults
/dev/sda1  /  ext4 defaults 1   1
```

To Exit and Save Changes Press: Esc

Type:- :wq

Press: Enter

Then we can configure this too:

Edit /etc/rc.conf to configure:

- Hostname
- Time zone
- Keyboard layout
- Services and modules

Run: vim /etc/rc.conf

```
root@localhost:~#
root@localhost:~# vim /etc/rc.conf
```

```
#

FONT=default
KEYMAP="us"
TIMEZONE="Africa/Addis_Ababa"
HOSTNAME="Abel"
LVM="no"
MODULES="ext4"
SERVICES="logger dhcpcd"

# End of file
~
~
~
~
~
~
~
~
~
~
~
~
~
~
~
~
~
~
~
~
~
~
~
~
"/etc/rc.conf" 13L, 177B written
root@localhost:~#
```

To Exit and Save Changes Press: Esc

Type:- :wq

Press: Enter

- ✓ **Then Set root password:**

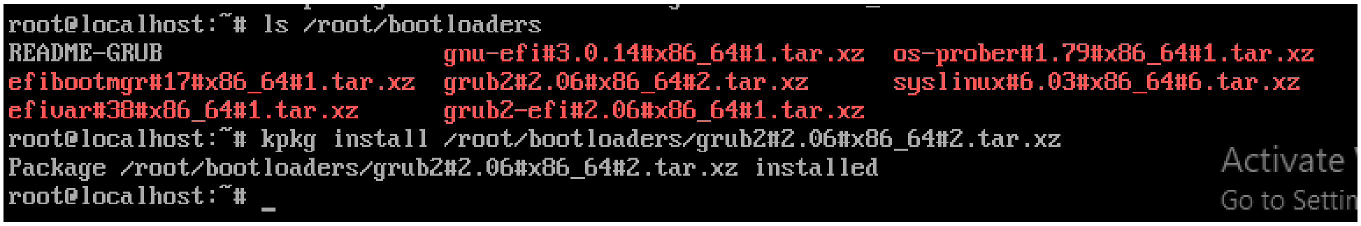
Run: passwd

```
root@localhost:~# passwd
New password:
Retype new password: _
```

Step 11: Install a Bootloader

First check the exact file name with ls /root/bootloaders | grep grub2

Then Run: kpkg install /root/bootloaders/grub2#24.2#x86_64#1.tar.xz



```
root@localhost:~# ls /root/bootloaders
README-GRUB                    gnu-efi#3.0.14#x86_64#1.tar.xz  os-prober#1.79#x86_64#1.tar.xz
efibootmgr#17#x86_64#1.tar.xz  grub2#2.06#x86_64#2.tar.xz      syslinux#6.03#x86_64#6.tar.xz
efivar#38#x86_64#1.tar.xz      grub2-efi#2.06#x86_64#1.tar.xz
root@localhost:~# kpkg install /root/bootloaders/grub2#2.06#x86_64#2.tar.xz
Package /root/bootloaders/grub2#2.06#x86_64#2.tar.xz installed
root@localhost:~# _
```

Step 12: **Generate the GRUB config:**

- ✓ **First creating grub directory**

Run: mkdir -p /boot/grub   then

Run: grub-mkconfig -o /boot/grub/grub.cfg

```
root@localhost:~# mkdir -p /boot/grub
root@localhost:~# grub-mkconfig -o /boot/grub/grub.cfg
Generating grub configuration file ...
Found linux image: /boot/vmlinuz-5.15.39
Warning: os-prober will not be executed to detect other bootable partitions.
Systems on them will not be added to the GRUB boot configuration.
Check GRUB_DISABLE_OS_PROBER documentation entry.
done
root@localhost:~# _
```

- ✓ **Then install grub**

Run: grub-install /dev/sda

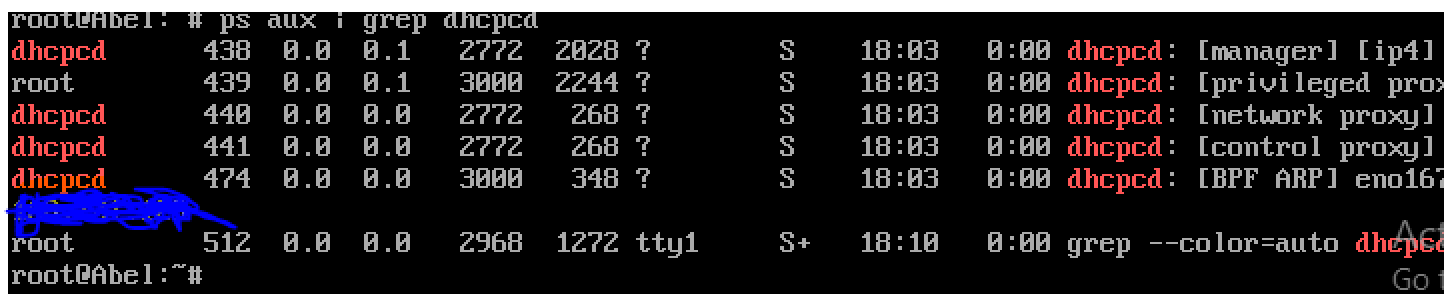
```
root@localhost:~# grub-install /dev/sda
Installing for i386-pc platform.
Installation finished. No error reported.
root@localhost:~# _
```

Step13:Run: reboot -f

Login & check everything:

- Can you log in?
- Is the network working? (Try `ping google.com`)
- Check installed services

Run: ps aux | grep dhcpcd



- ✓ **Login ✅**
- ✓ **that output shows dhcpcd is running correctly and has assigned an IP address to your network interface eno16777736. That means your internet connection is up and working in the virtual machine✅**

Issues I faced and solutions

➢ `kpkg install` didn't show output

- **Problem:** kpkg install didn't produce results or feedback after attempting to install packages like firefox, runit, xfce4, etc.
- **Cause:** Possible misconfigured package repositories or misunderstanding about available packages.
- **Solution:**
  - Verified existing package files (e.g., GRUB `.tar.xz`) and installed them locally using: kpkg install /path/to/package.tar.xz

➢ GRUB Configuration Failed

- **Problem:** `grub-mkconfig -o /boot/grub/grub.cfg` failed with error: `Directory nonexistent`.
- **Cause:** The `/boot/grub/` directory didn't exist.
- **Solution:**
  - Manually created the missing directory:

```
Using: mkdir -p /boot/grub
```

-
  - Then reran the GRUB configuration successfully.

➢ Missing Text Editor (nano)

- **Problem:** `nano` was not found.
- **Cause:** Minimal installation of Kwort doesn't include editors like `nano`.
- **Solution:**
  - Use `vi` as an alternative:
- Example vi /etc/resolv.conf

➢ Several typing mistakes and incorrect commands caused delays during the installation process.

Examples include using pkginstall instead of the correct command pkgsinstall , typing unmount instead of umount, and entering mdir instead of mkdir. These small errors led to confusion and required extra time to identify and correct, slowing down overall progress.

Filesystem Support

Used: ext4 (Fourth Extended Filesystem)

During the Kwort Linux installation, I used the `ext4` filesystem for the root partition (`/dev/sda1`). It is the most widely used filesystem in modern Linux distributions due to its performance, stability, and reliability.

Why ext4 Was Used

Advantages:

- Journaling (better data integrity)
- Widely supported in Linux
- Stable and fast

Not used:

- NTFS/FAT32 → Windows-centric, not native
- Btrfs/ZFS → More complex, not needed for basic setup

Advantages and Disadvantages of Kwort

Advantages	Disadvantages
✓ **Lightweight (fast boot)**	✓ **No GUI pre-installed**
✓ **Manual control (runit init)**	✓ **Smaller community/support**
✓ **Minimalist and transparent**	✓ **Requires more Linux knowledge**
✓ **Custom Package Manager (kpkg)**	✓ **Small User Base**
✓ **Good for Advanced Users & Developers**	✓ **Limited Package Availability**

Advantages	Disadvantages
	✓ **Not Ideal for Beginners** ✓ **No Official Desktop Environment**

Conclusion

The installation and configuration process of Kwort Linux provided hands-on exposure to the core components of a Linux operating system. Unlike user-friendly distributions, Kwort requires the user to engage directly with low-level system components, offering a deeper understanding of how Linux works under the hood.

Throughout the process, key skills were developed in areas such as partitioning, mounting filesystems, configuring system services, setting up networking with `dhcpcd`, and manually installing and configuring the GRUB2 bootloader. These tasks reinforced concepts like init systems (`runit`), network configuration, and the role of `rc.conf` in system initialization.

Additionally, troubleshooting real issues — such as lack of internet access despite DHCP processes running, or resolving GRUB installation errors due to missing directories — encouraged diagnostic thinking and problem-solving, which are critical skills in systems administration.

This experience not only strengthened my technical foundation in Linux system internals but also emphasized the importance of patience, precision, and understanding system feedback during installation. For anyone interested in system programming, network management, or DevOps, working with Kwort Linux is a highly educational journey that builds confidence and capability in managing Unix-like systems from the ground up.

Future Outlook / Recommendation

The successful installation of Kwort Linux opens doors to several future learning paths and opportunities for deeper exploration:

- **Explore Graphical User Interfaces (GUI):**
  After mastering the command-line interface, the next step is to install and configure a lightweight desktop environment such as **XFCE4**, along with a display manager like **LXDM** using:
  kpkg install xfce4 lxdm

This will provide practical experience in managing graphical sessions, user login interfaces, and system resources under a GUI.

- **Automate with Bash Scripting:**
Manual setup reinforces understanding, but automation is key in real-world system administration. Learning **bash scripting** will allow us to create custom installation scripts, automate network setups, manage services, and streamline repetitive tasks, making your work more efficient and error-free.
- **Experiment with More Linux Distributions:**
Each Linux distribution offers unique philosophies and challenges. **Arch Linux** promotes DIY configuration with a rolling release model, while **Alpine Linux** is security-focused and extremely lightweight. Exploring these will broaden our understanding of system design choices, package management, and community philosophies.
- **Try Other Hypervisors:**
While VMware is a powerful virtualization tool, experimenting with alternatives like **VirtualBox** (GUI-focused) and **QEMU** (more CLI and scripting-oriented) will enhance our virtualization skills. Each tool has its advantages, and understanding how they differ helps in selecting the right tool for specific tasks.
- **Deepen our Knowledge of System Internals:**
With a base understanding now in place, continue exploring topics such as **init systems (runit vs. systemd)**, **kernel module management**, **bootloaders (GRUB vs. LILO)**, and **network configurations** at the low level. We can even try customizing our own Linux build using tools like Linux From Scratch (LFS).
- **Contribute or Document:**
As we gain confidence, consider writing guides, documenting our learning journey, or contributing to open-source projects. It strengthens our understanding and helps others in the community.

Virtualization: What, Why, and How

What: Virtualization allows multiple OSes to run on a single physical machine using virtual machines (VMs).

Why:

- Efficient resource usage
- Testing and isolation
- Easy backup and restore
- Security sandboxing

How:

- Hypervisors like VMware/VirtualBox manage VMs
- Each VM has virtualized CPU, RAM, disk
- OS inside VM thinks it runs on real hardware

Implement System Calls

To better understand how the kernel interacts with user-level programs, I implemented a simple C program using the execv() system call. This function is part of the unistd.h library in Unix-like systems and replaces the current process image with a new one — typically another executable.

Source File:exec_test.c

```
#include <unistd.h>

#include <stdio.h>



int main() {

    char *args[] = {"/bin/ls", "-l", NULL};

    execv(args[0], args);

    perror("exec failed");

    return 1;

}
```

How it works:

Explanation:

- **Execv (path, args): This system call executes the program specified in path (here, /bin/ls) with the arguments in args.**
- **If successful, the current process is replaced and does not return to the caller.**
- **If it fails, perror() displays the error.**

Compilation & Execution

:To compile: Run : vi exec_test.c

Next type i in order to be able to write in the compiler now we can write the code

```
#include <stdio.h>
#include <unistd.h>
int main() {
char *args[] = {"/bin/ls", "-1", NULL};
execvp(args[0],args);
perror("exec failed");
return 1;
}
~
~
~
~
~
~
~
~
~
~
~
~
~
~
~
~
~
```

Then

To Exit and Save Changes Press: Esc

Type :-   :wq

Press: Enter

Finally RUN: gcc exec_test.c -o exec_test

Then ./test

```
root@Abel:~# gcc exec_test.c -o test
root@Abel:~# ./test
bootloaders
exec_example1.c
exec_test.c
test
root@Abel:~#
```

What the exec() System Call Did:

My C program (exec_test.c) called exec() (e.g., execlp() or execvp()) to **replace the current process image** with the ls command. That means:

- The test process was started.
- It immediately used exec() to run ls, effectively replacing itself with the ls process.
- Therefore, you see the output of ls instead of anything from the original C code that might have followed the exec() call.

Summary:

The exec() system call **replaces the current process** with a new one — in this case, ls. My program successfully demonstrated this behavior by printing the contents of the directory and exiting without returning to the original C process logic.