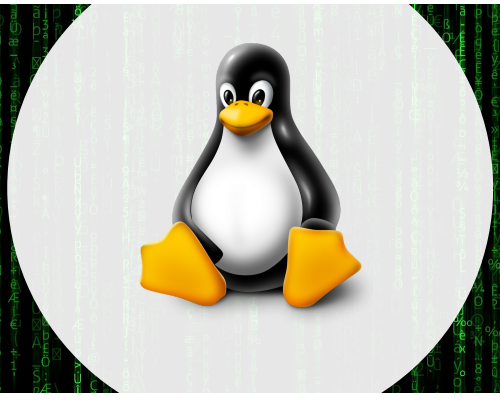
Linux Kernel Building Guide



PREPARED BY :

ABHISHEK PS & JISHNU DEVADAS

CSE A

OVERVIEW

Building a Linux kernel is a seemingly complex yet relatively simple task.

In this documentation we will dive into the requirements, procedure as well as installation of a new Linux kernel built by yourself .

When compared to an OS like Windows , Linux is far more customizable depending on the user's liking which is also due to it's open-source nature.

A huge part of why Linux is so flexible is because of its modular structure. Every single system component and program is split up into many different packages that can easily be removed, added, or replaced by something else.

Absolutely no part of the operating system has an exception to that rule, including the Linux kernel itself.

All Linux distributions are based on a predefined kernel. But, if you want to disable certain options and drivers or try experimental patches, you need to compile your own Linux kernel.

Now we will look into the requirements.

REQUIREMENTS

Even though there is not a specific system requirement for building a Linux kernel on your PC/Laptop , having met the minimum prerequisites is good as it helps in speeding up the process and run less into problems.

- Internet Connection for downloading source code and libraries/packages
- A system running Linux
- Access to the terminal/command line

- A user account with sudo/root privileges
- 12GB of available space on the hard drive

It is also safer to do this in a Virtual Machine but it is not a requirement in case you can't run a VM on your PC smoothly.

STEPS TO BUILD THE LINUX KERNEL

1. Download Source Code

You can download the source code from https://kernel.org

OR

You can do the same by opening your terminal and type

wget (followed by the link to the kernel source code by right clicking the yellow button and clicking copy link address and pasting it into the terminal)

```
prozpekt@prozpekt:~$ wget https://cdn.kernel.org/pub/linux/kernel/v6.x/linux-6.2.10.tar.xz
```

```
prozpekt@prozpekt:~$ wget https://cdn.kernel.org/pub/linux/kernel/v6.x/linux-6.2.10.tar.xz
--2023-04-12 16:01:11--  https://cdn.kernel.org/pub/linux/kernel/v6.x/linux-6.2.10.tar.xz
Resolving cdn.kernel.org (cdn.kernel.org)...  199.232.45.176, 2a04:4e42:48::432
```

```
Connecting to cdn.kernel.org (cdn.kernel.org)|199.232.45.176|:443
... connected.
HTTP request sent, awaiting response... 200 OK
Length: 136456416 (130M) [application/x-xz]
Saving to: 'linux-6.2.10.tar.xz'

linux-6.2.10.tar 100%[==========>] 130.13M  51.9MB/s    in 2.5s

2023-04-12 16:01:14 (51.9 MB/s) - 'linux-6.2.10.tar.xz' saved [13
6456416/136456416]
```

Now hit enter and it should start downloading the source code to your PC.

2. Extract the tar file

```
prozpekt@prozpekt:~$ tar xf linux-6.2.10.tar.xz
prozpekt@prozpekt:~$ ls
bin  crDroid  crDroid_gsi  linux-6.2.10  linux-6.2.10.tar.xz
```

Use ls **to list all files in your directory.**

Find the .xz **file and run the command**

tar xf (followed by file name)

3. Install necessary packages for compiling the kernel

sudo apt-get install git fakeroot build-essential ncurses-dev xz-utils libssl-dev bc flex libelf-dev bison

Package	Package description
git	**Tracks and makes a record of all changes during development in the source code. It also allows reverting the changes.**
fakeroot	**Creates the fake root environment.**
build-essential	**Installs development tools such as C, C++, gcc, and g++.**
ncurses-dev	**Provides API for the text-based terminals.**
xz-utils	**Provides fast** file compression and decompression.

libssl-dev	**Supports** SSL and TSL that encrypt data and make the internet connection secure.
bc (Basic Calculator)	**Supports the interactive execution of statements.**
flex (Fast Lexical Analyzer Generator)	**Generates lexical analyzers that convert characters into tokens.**
libelf-dev	**Issues a shared library for managing ELF files (executable files, core dumps and object code)**
bison	**Converts grammar description to a C program.**

4. Configure the kernel

Now 'cd' **into the extracted file.**

```
prozpekt@prozpekt:~$ cd linux-6.2.10
prozpekt@prozpekt:~/linux-6.2.10$ ls
COPYING          certs                    net
CREDITS          crypto                   rust
Documentation    drivers                  samples
Kbuild           fs                       scripts
Kconfig          include                  security
LICENSES         init                     sound
MAINTAINERS      io_uring                 tools
Makefile         ipc                      usr
Module.symvers   kernel                   virt
README           lib                      vmlinux
System.map       mm                       vmlinux-gdb.py
arch             modules.builtin          vmlinux.a
block            modules.builtin.modinfo  vmlinux.map
built-in.a       modules.order            vmlinux.o
```

And type

```
built-in.a       modules.order            vmlinux.o
prozpekt@prozpekt:~/linux-6.2.10$ cp /boot/config-$(uname -r) .co
nfig
```

cp /boot/config-$(uname -r) .config

This is done to copy the existing configuration file.

Then

```
prozpekt@prozpekt:~/linux-6.2.10$ make olddefconfig
.config:428:warning: symbol value 'm' invalid for I8K
.config:1922:warning: symbol value 'm' invalid for MCTP
.config:6057:warning: symbol value 'm' invalid for ANDROID_BINDER
_IPC
.config:6058:warning: symbol value 'm' invalid for ANDROID_BINDER
FS
#
# configuration written to .config
#
```

type 'make olddefconfig'

This is used to auto answer 'default' for new options to be setup in the kernel configuration and also keeps all the old options from 'old.config' file. Certain warnings can be ignored. As long it doesn't display errors you're good to go.

4. Next disable trusted keys

This is specific to Debian based distros which in turn includes Ubuntu based distros. So if you're running Debian , Ubuntu , Mint , Kali , Pop!_OS etc you might need to do the following

```
# configuration written to .config
#
prozpekt@prozpekt:~/linux-6.2.10$ scripts/config --disable SYSTEM_TRUSTED_KEYS &
& scripts/config --disable SYSTEM_REVOCATION_KEYS
```

scripts/config --disable SYSTEM_TRUSTED_KEYS && scripts/config --disable SYSTEM_REVOCATION_KEYS

And hit enter

5. Compiling the kernel

Now we have reached the important part of the process which is the actual compilation of the kernel.

To do this type

```
prozpekt@prozpekt:~/linux-6.2.10$ scripts/config --disable SYSTEM_TRUSTED_KEYS &
& scripts/config --disable SYSTEM_REVOCATION_KEYS
prozpekt@prozpekt:~/linux-6.2.10$ make -j$(nproc --all)
```

make -j$(nproc --all)

OR

The j value can also be entered manually instead of auto fetching , then the code would be

make -jX

where X is the no of threads in the CPU/ installed processing units. In this case we used an AMD EPYC 7002 Server CPU which has 256 threads so typing

make -j256

Would do the same but it depends on the number of your CPU threads.

Hit enter and now the compilation process shall start and hopefully you would not come across errors. Most errors would be missing libraries or memory being full which should be addressed accordingly.

Time taken to compile is also relative to your PC's hardware capabilities. It might takes a few hours.

After compiling if no errors are displayed then the compilation went through successfully.

6. Installing the modules and kernels

```
prozpekt@prozpekt:~/linux-6.2.10$ sudo make modules_install && sudo make install
```

Then type

sudo make models_install && sudo make install

This is to install the modules and the kernel.

It also automates certain commands

```
  INSTALL /lib/modules/6.2.10/kernel/net/hsr/hsr.ko
  SIGN    /lib/modules/6.2.10/kernel/net/hsr/hsr.ko
  INSTALL /lib/modules/6.2.10/kernel/net/qrtr/qrtr.ko
  SIGN    /lib/modules/6.2.10/kernel/net/qrtr/qrtr.ko
  INSTALL /lib/modules/6.2.10/kernel/net/qrtr/qrtr-smd.ko
  SIGN    /lib/modules/6.2.10/kernel/net/qrtr/qrtr-smd.ko
  INSTALL /lib/modules/6.2.10/kernel/net/qrtr/qrtr-tun.ko
  SIGN    /lib/modules/6.2.10/kernel/net/qrtr/qrtr-tun.ko
  INSTALL /lib/modules/6.2.10/kernel/net/qrtr/qrtr-mhi.ko
  SIGN    /lib/modules/6.2.10/kernel/net/qrtr/qrtr-mhi.ko
  DEPMOD  /lib/modules/6.2.10
  INSTALL /boot
run-parts: executing /etc/kernel/postinst.d/initramfs-tools 6.2.10 /b
oot/vmlinuz-6.2.10
update-initramfs: Generating /boot/initrd.img-6.2.10
I: The initramfs will attempt to resume from /dev/sda2
I: (UUID=1e2f4dad-43b0-4dea-8af5-5b98f2a51454)
I: Set the RESUME variable to override this.
run-parts: executing /etc/kernel/postinst.d/unattended-upgrades 6.2.1
0 /boot/vmlinuz-6.2.10
run-parts: executing /etc/kernel/postinst.d/update-notifier 6.2.10 /b
oot/vmlinuz-6.2.10
run-parts: executing /etc/kernel/postinst.d/xx-update-initrd-links 6.
```

```
2.10 /boot/vmlinuz-6.2.10
I: /boot/initrd.img.old is now a symlink to initrd.img-5.15.0-1035-az
ure
I: /boot/initrd.img is now a symlink to initrd.img-6.2.10
run-parts: executing /etc/kernel/postinst.d/zz-shim 6.2.10 /boot/vmli
nuz-6.2.10
run-parts: executing /etc/kernel/postinst.d/zz-update-grub 6.2.10 /bo
ot/vmlinuz-6.2.10
Sourcing file `/etc/default/grub'
Sourcing file `/etc/default/grub.d/40-force-partuuid.cfg'
Sourcing file `/etc/default/grub.d/50-cloudimg-settings.cfg'
Sourcing file `/etc/default/grub.d/init-select.cfg'
Generating grub configuration file ...
GRUB_FORCE_PARTUUID is set, will attempt initrdless boot
Found linux image: /boot/vmlinuz-6.2.10
Found initrd image: /boot/initrd.img-6.2.10
Found linux image: /boot/vmlinuz-5.15.0-1035-azure
Found initrd image: /boot/initrd.img-5.15.0-1035-azure
Found linux image: /boot/vmlinuz-5.15.0-1034-azure
Found initrd image: /boot/initrd.img-5.15.0-1034-azure
Warning: os-prober will not be executed to detect other bootable part
itions.
Systems on them will not be added to the GRUB boot configuration.
Check GRUB_DISABLE_OS_PROBER documentation entry.
Adding boot menu entry for UEFI Firmware Settings ...
done
prozpekt@prozpekt:~/linux-6.2.10$
```

7. Reboot into new kernel

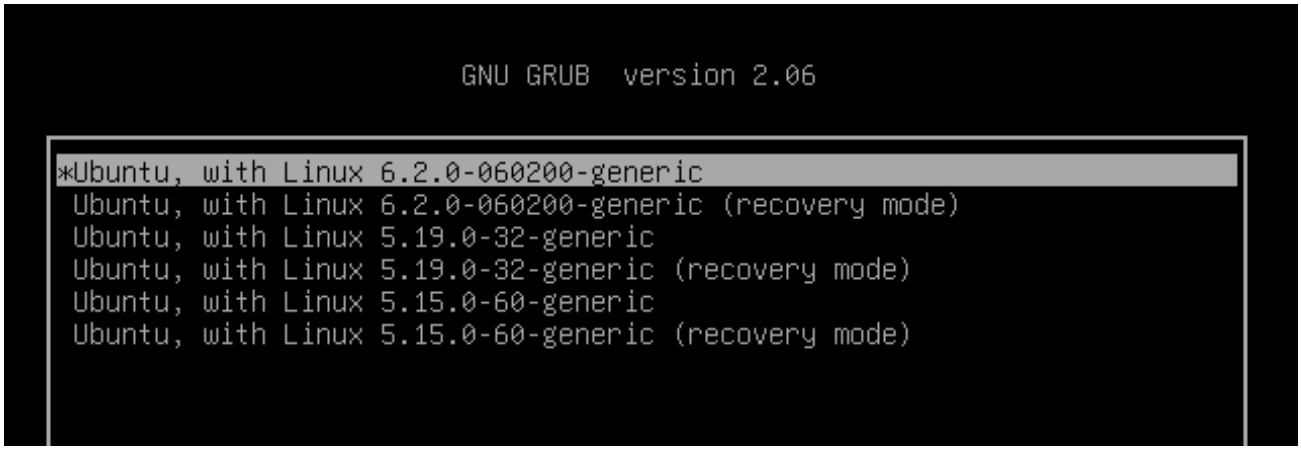
After installation you can now reboot the PC as usual through GUI or through command line by

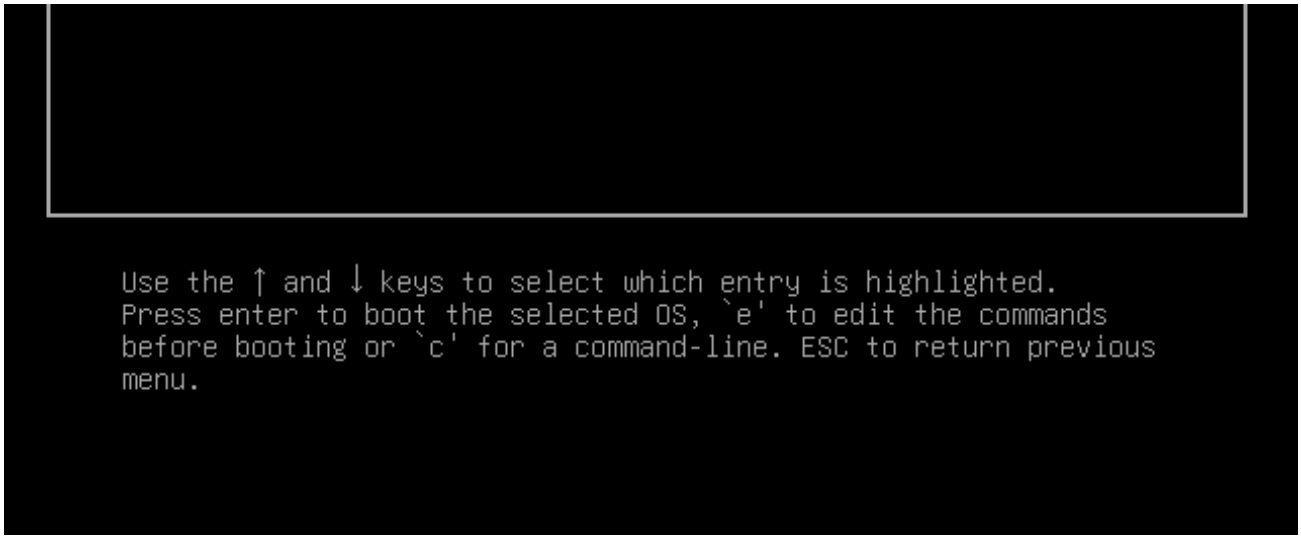
sudo reboot

OR

sudo shutdown -r now

Now you have successfully installed the new kernel and can select it from the GRUB bootloader by looking at the kernel version number.





After rebooting into new kernel you can run the command

uname -r

to check the kernel version.

NOTE :

- **In case the kernel does not boot you can boot into the old kernel by selecting it from the bootloader(GRUB).**
- **You can also check the purpose of each command by using man command.**

The following are not required to be done but can be done with caution

TO CONFIGURE THE KERNEL IN AN ADVANCED MANNER

- **Instead of olddefconfig , you can use menuconfig for a text based kernel configuration with the option to configure every aspect of your kernel . xconfig can be used for doing the same but in GUI . These alternatives must be used with caution as it can either make or break your kernel. There are many options which include critical components of the kernel better to be left untouched.**

TO ADD A CUSTOM KERNEL NAME AS SUFFIX

- **If you want a custom kernel name as extension to the 'vmlinuz' file which is the actual kernel , then you can use menuconfig after running olddefconfig and before compiling the kernel . Go into General Setup -> Local Version and add a suffix. Hence if we add 'hello' as suffix the kernel would be named 'vmlinuz-(ver no.)-hello' .**

To fetch, build and install the latest Linux kernel to a specific Linux distribution through automation use KernelWiz

Link : https://github.com/psabhishek26/KernelWiz