**Linux kernel compilation**

The [Linux Kernel](https://phoenixnap.com/glossary/what-is-a-linux-kernel) is the foundation of all the Linux distributions. [The kernel](https://phoenixnap.com/glossary/what-is-a-kernel) is responsible for communication between hardware and software and the allocation of available resources.

All [Linux](https://phoenixnap.com/glossary/what-is-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.

**In this step-by-step guide, you will learn how to build and compile a Linux kernel from scratch.**

**Prerequisites**

* A system running Linux
* Access to the terminal/command line
* A user account with **sudo/root** privileges
* 12GB of [available space](https://phoenixnap.com/kb/linux-check-disk-space) on the hard drive

## Building Linux Kernel

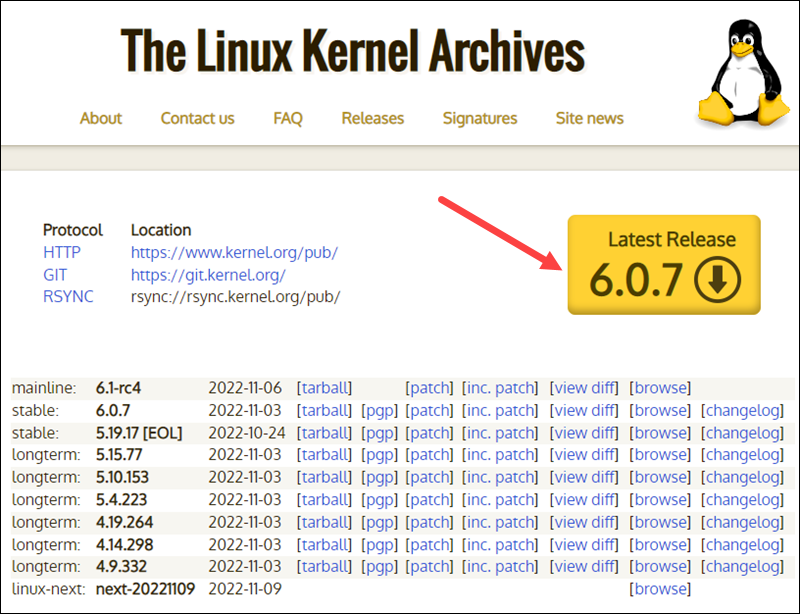
The process of building a Linux kernel can be performed in seven easy steps. However, the procedure may require a significant amount of time to complete, depending on the system speed.

Follow the steps below to build the latest Linux kernel.

**Note:** If the latest kernel version on the official website does not match the one mentioned in the steps below, replace the version number in the commands with the current latest version.

### Step 1: Download the Source Code

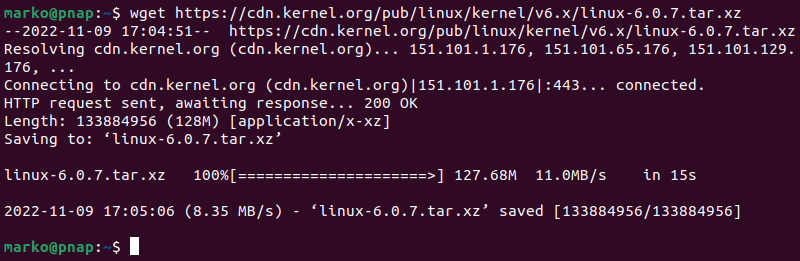
1. Visit the [official kernel website](https://www.kernel.org/) and download the [latest kernel version](https://phoenixnap.com/kb/check-linux-kernel-version). The downloaded file contains a compressed [source code](https://phoenixnap.com/glossary/what-is-source-code).



2. Open the terminal and use the [wget command](https://phoenixnap.com/kb/wget-command-with-examples" \t "_blank) to download the Linux kernel source code:

wget https://cdn.kernel.org/pub/linux/kernel/v6.x/linux-6.0.7.tar.xz

The output shows the “saved” message when the download completes.



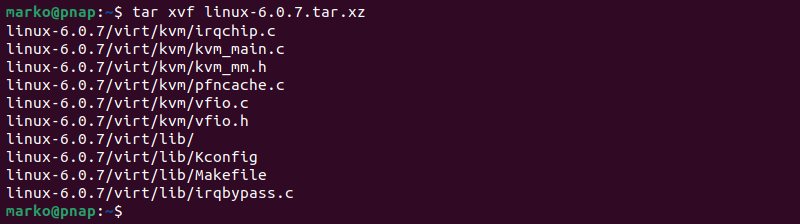
**Note:** Learn what to do when you get [wget: command not found](https://phoenixnap.com/kb/wget-command-not-found" \t "_blank) error.

### Step 2: Extract the Source Code

When the file is ready, [run the tar command](https://phoenixnap.com/kb/tar-command-in-linux) to extract the source code:

tar xvf linux-6.0.7.tar.xz

The output displays the extracted kernel source code:



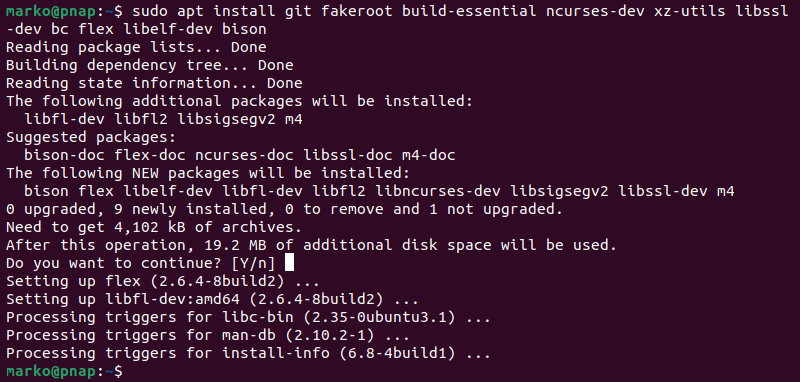
### Step 3: Install Required Packages

Install additional packages before building a kernel. To do so, run this command:

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

The command we used above installs the following packages:

|  |  |
| --- | --- |
| **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](https://phoenixnap.com/glossary/c-programming-language), [C++](https://phoenixnap.com/glossary/c-plus-plus-programming-language), gcc, and g++. |
| **ncurses-dev** | Provides API for the text-based terminals. |
| **xz-utils** | Provides fast [file compression](https://phoenixnap.com/glossary/file-compression) and decompression. |
| **libssl-dev** | Supports [SSL and TSL](https://phoenixnap.com/kb/tls-vs-ssl) 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. |



### Step 4: Configure Kernel

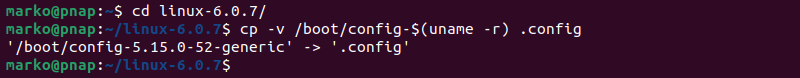
The Linux kernel source code comes with the default configuration. However, you can adjust it to your needs. To do so, follow the steps below:

1. Navigate to the **linux-6.0.7** directory using the [cd command](https://phoenixnap.com/kb/linux-cd-command):

cd linux-6.0.7

2. Copy the existing configuration file using the **cp** command:

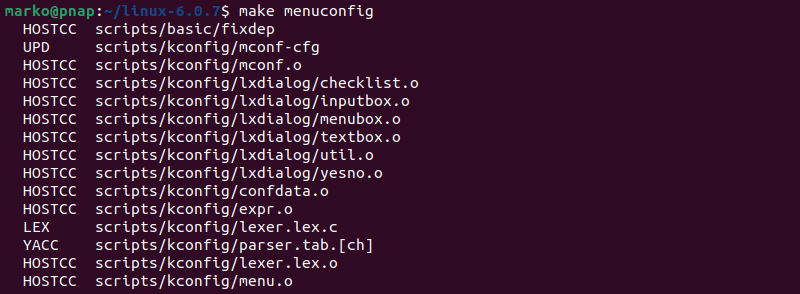
cp -v /boot/config-$(uname -r) .config



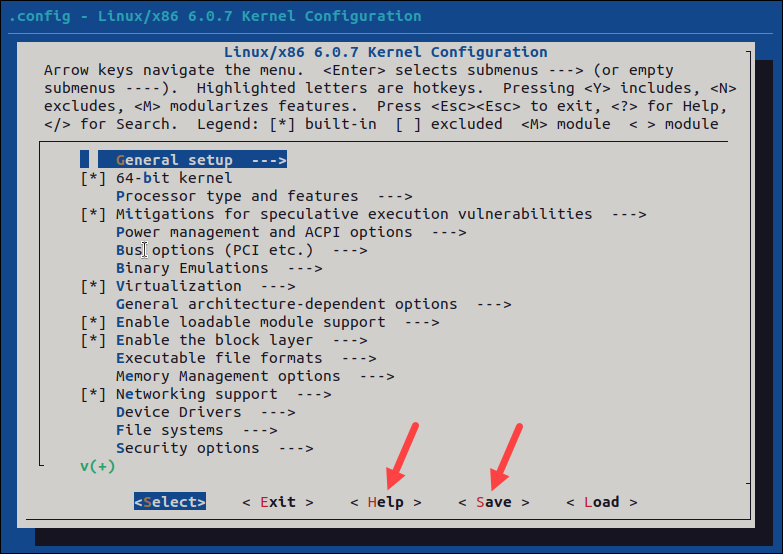
3. To make changes to the configuration file, run the [make command](https://phoenixnap.com/kb/linux-make-command):

make menuconfig

The command launches several scripts that open the configuration menu:



4. The configuration menu includes options such as [firmware](https://phoenixnap.com/glossary/firmware), file system, network, and memory settings. Use the arrows to make a selection or choose **Help** to learn more about the options. When you finish making the changes, select **Save,** and then exit the menu.



**Note:**Changing settings for some options can lead to a non-functional kernel. If you are unsure what to change, leave the default settings.

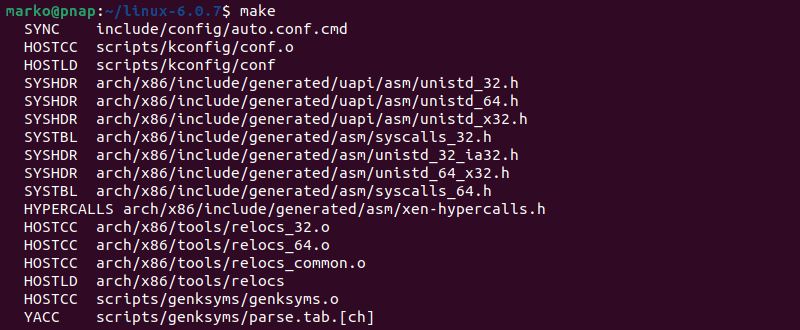
### Step 5: Build the Kernel

1. Start building the kernel by running the following command:

make

The process of building and compiling the Linux kernel takes some time to complete.

The terminal lists all Linux kernel components: [memory management](https://phoenixnap.com/glossary/memory-management), hardware device drivers, filesystem drivers, network drivers, and process management.



If you are compiling the kernel on Ubuntu, you may receive the following error that interrupts the building process:

No rule to make target 'debian/canonical-certs.pem

Disable the conflicting security certificates by executing the two commands below:

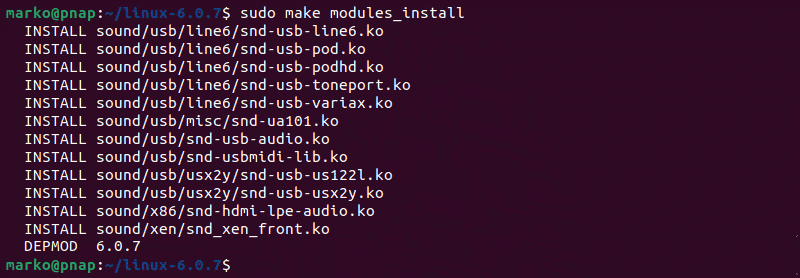
scripts/config --disable SYSTEM\_TRUSTED\_KEYS

scripts/config --disable SYSTEM\_REVOCATION\_KEYS

The commands return no output. Start the building process again with **make**, and press **Enter** repeatedly to confirm the default options for the generation of new certificates.

2. Install the required modules with this command:

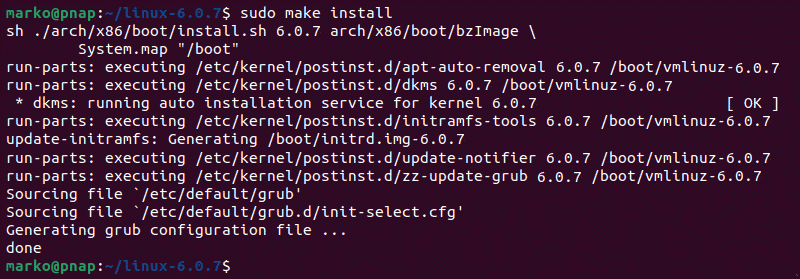
sudo make modules\_install



3. Finally, install the kernel by typing:

sudo make install

The output shows **done** when finished:



### Step 6: Update the Bootloader (Optional)

The GRUB bootloader is the first program that runs when the system powers on.

The **make install** command performs this process automatically, but you can also do it manually.

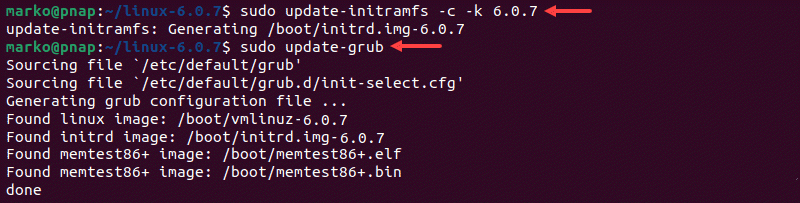
1. Update the **initramfs** to the installed kernel version:

sudo update-initramfs -c -k 6.0.7

2. Update the **GRUB bootloader** with this command:

sudo update-grub

The terminal prints out the process and confirmation message:



### Step 7: Reboot and Verify Kernel Version

When you complete the steps above, reboot the machine.

When the system boots up, verify the kernel version using the [uname](https://phoenixnap.com/kb/uname-linux" \t "_blank) command:

uname -mrs

The terminal prints out the current Linux kernel version.

Verify the current Linux Kernel version.

**Conclusion**

In this step-by-step guide, you learned how to build a Linux kernel from scratch and install the required packages.

If you follow the instructions carefully, the process will complete successfully on your Linux machine.

The Linux kernel has a modular design. Functionality is extendible with modules or drivers. Learn how to use the [modprobe command](https://phoenixnap.com/kb/modprobe-command" \t "_blank) to add or remove modules on Linux.

**Introduction**

The [Linux kernel](https://phoenixnap.com/glossary/what-is-a-linux-kernel) has a modular design. Functionality is extendible with modules or drivers. Use the **modprobe** command to add or remove modules on Linux. The command works intelligently and adds any dependent modules automatically.

The [kernel](https://phoenixnap.com/glossary/what-is-a-kernel) uses **modprobe** to request modules. The **modprobe** command searches through the standard installed module directories to find the necessary drivers.

**Prerequisites**

* A system running Linux
* Access to the terminal/command line
* A user account with **sudo**or **root** privileges

## modprobe Command Syntax Explained

All **modprobe** commands require sudo privileges. The general syntax for using **modprobe** is:

sudo modprobe <options> <module name>

By default, the **modprobe** command adds a module.

For multiple modules, expand with the option **-a** or **-all**:

sudo modprobe <options> -a <first module name> <second module name>

## Options for modprobe Command

The available **modprobe** command options are divided into categories based on their use-case.

### Management Options

Management options enable different module handling situations when **inserting or removing** modules with the **modprobe** command.

|  |  |
| --- | --- |
| **--all** **-a** | Enables multiple modules to be inserted or removed at the same time. |
| **--remove** **-r** | Remove a module. Applies **--remove-dependencies** as well. Useful for removing  broken modules. |
| **--remove-dependencies** | Removes dependency modules. |
| **--resolve-alias** **-R** | Look up and print all module names matching an alias.  Useful for debugging alias problems. |
| **--first-time** | Prints an error for already inserted or removed modules. |
| **--ignore-install** **--ignore-remove** **-i** | Ignore install/remove commands written in the module  when inserting/removing a module. |
| **--use-blacklist** **-b** | Blacklist resolved alias. Blacklisted modules are not automatically loaded. |
| **--force** **-f** | Force module insertion or removal when version errors appear.  Applies both **--force-modversion** and **--force-vermagic**. Use with caution. |
| **--force-modversion** | Ignore module version on insertion or removal. Use with caution. |
| **--force-vermagic** | Ignore module version magic on insertion or removal. Use with caution. |

### Query Options

Query options for **modprobe** show information about configuration and dependencies.

|  |  |
| --- | --- |
| **--show-depends** **-D** | Lists the module with the dependency files if there are any.  The dependencies that install with the module have the “install” prefix. |
| **--showconfig** **--show-config** **-c** | Prints current configuration and exists. |
| **--show-modversions** **--dump-modversions** | Dumps module version dependencies. |

### General Options

General options configure **modprobe** output options, module locations, and versions.

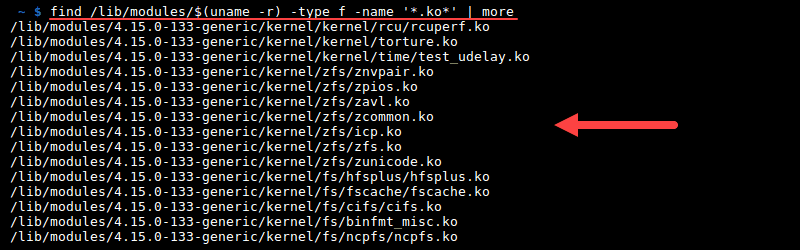
|  |  |
| --- | --- |
| **--dry-run** **--show** **-n** | Do not execute insert/remove but print the output. Used for debugging purposes. |
| **--config=<file name>** **-C** | Overrides the default configuration dependency (/etc/modprobe.d)  with <file name>. |
| **--dirname=<directory>** **-d** | Use <directory> as filesystem root for /lib/modules. |
| **--set-version=<version>** **-S** | Use specified kernel <version> instead of using **uname**. |
| **--syslog** **-s** | Prints the error messages through syslog instead of standard error (stderr).  When stderr is unavailable, errors get printed to syslog automatically. |
| **--quiet** **-q** | Disables display of error messages. |
| **--verbose** **-v** | Enables more messages to show, if available. **modprobe** only  prints messages if something goes wrong. |
| **--version** **-V** | Shows the **modprobe** version. |
| **--help** **-h** | Shows the help message with all the commands listed. |

## Examples of modprobe Command

All kernel modules are listed in the /lib/modules directory system in .ko (kernel object) files by default.

Find all the available modules for the current [kernel version](https://phoenixnap.com/kb/check-linux-kernel-version) with:

find /lib/modules/$(uname -r) -type f -name '\*.ko\*' | more



**Note:** Consider removing old kernel versions. Check out our guide on [how to remove old kernels on Ubuntu](https://phoenixnap.com/kb/ubuntu-remove-old-kernels).

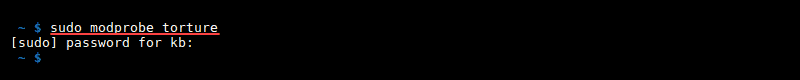
### Adding Kernel Modules With modprobe Command

1. Add a module using the **modprobe** command:

sudo modprobe <module name>

For example:

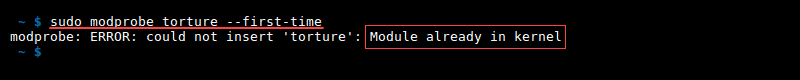
sudo modprobe torture



2. Confirm the module loaded with:

sudo modprobe <module name> --first-time

The output prints an error because the module is already in the kernel.

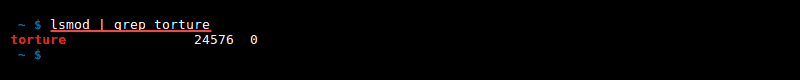


Alternatively, find the module in the active module loaded list with **lmod**:

lsmod | grep <module name>

For example:

lsmod | grep torture



### Removing Kernel Modules With modprobe Command

1. Remove a module using the **modprobe -r** command:

sudo modprobe -r <module name>

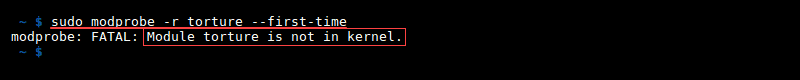
For example:

sudo modprobe -r torture

2. Confirm the module is removed by running:

sudo modprobe -r <module name> --first-time

An error message appears saying that the module is not in the kernel:



Alternatively, check the active module loaded list:

lsmod | grep <module name>

The removed module is not on the module loaded list.

**Conclusion**

The Linux kernel is created to be modular and easily extendible. Make sure to research the modules you want to add or remove to avoid problems with the kernel.