Setting Up and Configuring the latest linux kernel

Welcome! This is the documentation of the steps I followed to successfully compile a kernel. Before actually diving into the kernel install, check the version which you are originally running using the following command.

```
$ uname -r
4.4.0-116-generic
```

The command shows the current version of the kernel, which on my system is `4.4.0-116-generic`. At the time of writing the documentation, the latest version is `4.16.13`. The latest version for the kernel can be checked here

Once you have decided onto the version of the kernel you want to build, next step is to download the kernel. I used `wget` for the same, but there are other ways as well.

Type the following command in your terminal:

```
$ wget https://cdn.kernel.org/pub/linux/kernel/v4.x/linux-4.16.13.tar.xz
```

The following command will download a `tar.xz` archives for you. To extract the files, use the following command:

```
$ tar -xf linux-4.16.13.tar.xz
```

The command will extract the contents of the tar file.

NOTE: You will have to write your own version instead of 4.16.13.

NOTE: To install the `net-next`*, I used* `git clone` *instead of getting the kernel folder. The command for same is given below. Also note that you are not required to go through the extract the archive steps.*

```
git clone https://git.kernel.org/pub/scm/linux/kernel/git/davem/net-next.git
```

This might be a huge file and if you are working with the kernel for experimental purposes and do not intend to work as a Linux Kernel Developer, you might want to run it with the `--depth=1` option.

```
git clone --depth=1 https://git.kernel.org/pub/scm/linux/kernel/git/davem/net-next.git
```

The `--depth` option is used with `git clone` to create a shallow clone with its history truncated.

NOTE: Before proceeding make sure you have enough space in your disk to complete the kernel build.

To know the space in various partitions, do

```
$ df -h
```

You will get results like these:

```
Filesystem                                  Size  Used Avail Use% Mounted on
udev                                         32G     0   32G   0% /dev
tmpfs                                       6.3G  9.0M  6.3G   1% /run
/dev/nvme0n1p1                               16G  3.2G   12G  22% /
tmpfs                                        32G     0   32G   0% /dev/shm
tmpfs                                       5.0M     0  5.0M   0% /run/lock
tmpfs                                        32G     0   32G   0% /sys/fs/cgroup
ops.utah.cloudlab.us:/proj/cloudlab-PG0     100G   24G   77G  24% /proj/cloudlab-PG0
ops.utah.cloudlab.us:/share                  97G   10G   80G  12% /share
tmpfs                                       6.3G     0  6.3G   0% /run/user/20001
```

On an average the kernel build requires much more space than its actual size. The size of the kernel can be found out by `du -ch`.

```
$ cd linux-4.16.13
$ du -ch
919M
```

The command `du` will tell you about the size of the folder in a format depending on the flags specified. The `h` flag will produce the result in *human-readable* format (e.g. 200K, 960M). The `c` flag gives the grand-total of all the contents of the folder. For more info on the options available to the command, do `man du`.

In my case, it was 919M. Also note that the kernel size will vary based on the drivers and configuration you choose for your kernel, the command for which is explained later. It is recommended to make proper space (at least 4GB) in your disk before building the kernel.

The steps from now focus on building the kernel.

```
$ cd /linux-4.16.13
```

The next task in hand is to configure the kernel to tell it about the modules to be included.

```
$ sudo make menuconfig
```

The command will open a TUI (Text-Based UI) for the configuration, with directions on the top to select/de-select.I chose the default configuration for now. After you are done with choosing your configuration, save the configuration and exit. This generates a `.config` file.

NOTE: It is recommended to make a copy of your existing kernel before making any changes.

Now we move on to compiling the kernel and the kernel modules. The command for doing the same is `make`. Before proceeding I checked the number of processors available to the current process to speed up the process by using the linux command `nproc`.

```
$ nproc
16
```

```
$ sudo make -j 16
```

Now compile the kernel modules using the following command:

```
$ sudo make modules_install -j 16
```

Now the .config file and the kernel needs to be copied to the `/boot`folder along with the generation of the system.map file.

```
$ sudo make install -j 16
```

We have compiled the kernel successfully. Now to let the system use the new kernel when it boots up next, we need to update the grub.

```
$ sudo update-initramfs -c -k 4.16.13
$ sudo update-grub
```

NOTE: While building the net-next, the version is `4.17.0-rc+`*. Make sure to use the right kernel version which can also be known from the last command used i.e* `sudo make install`, the initial output for which is given here.

```
sh ./arch/x86/boot/install.sh 4.17.0-rc7+ arch/x86/boot/bzImage \
System.map "/boot"
```

You can check the version here. [NOTE ENDS HERE]

After the above commands run successfully, reboot the system to start working with the new kernel.

You can always verify the kernel with `uname -r`.

```
$ uname -r
4.16.13
```

This shows that the kernel has been successfully compiled and running.

Cheers!!

Credits and Sources

1. Archlinux Kernel/Traditional Compilation
2. FreeCodeCamp Article
3. Tedfelix Article
4. Git Clone
5. Fraida Fund, NYU