Installation on Linux

This chapter describes how to install `libfranka` and `franka_ros`, either as binary packages or by building from source, and how to install a real-time Linux kernel. `franka_ros` is only required if you want to control your robot using ROS.

Note

While `libfranka` and the `franka_ros` packages should work on different Linux distributions, official support is currently only provided for:

- Ubuntu 18.04 LTS *Bionic Beaver* and ROS *Melodic Morenia* (requires at least `libfranka` 0.6.0)
- Ubuntu 20.04 LTS *Focal Fossa* and ROS *Noetic Ninjemys* (requires at least `libfranka` 0.8.0)

The following instructions are exemplary for Ubuntu 20.04 LTS system and ROS *Noetic Ninjemys*. They only work in the supported environments.

Warning

We do not offer support for Ubuntu 16.04 LTS *Xenial Xerus* and ROS *Kinetic Kame* anymore, as they have reached their end-of-life.

Installing from the ROS repositories

Hint

These packages might not always be up-to-date, as they are only synced at certain intervals. Read the changelog at https://frankaemika.github.io to find out which `libfranka` version is required for a particular robot software version. If this doesn't match the `ros-noetic-libfranka` version from the repositories, you need to build from source.

Binary packages for `libfranka` and `franka_ros` are available from the ROS repositories. After setting up ROS Noetic, execute:

```
sudo apt install ros-noetic-libfranka ros-noetic-franka-ros
```

Building from source

Refer to the README.md

Building the ROS packages

After setting up ROS Noetic, create a Catkin workspace in a directory of your choice:

```
cd /path/to/desired/folder
mkdir -p catkin_ws/src
cd catkin_ws
source /opt/ros/noetic/setup.sh
catkin_init_workspace src
```

Then clone the `franka_ros` repository from GitHub:

```
git clone --recursive https://github.com/frankaemika/franka_ros src/franka_ros
```

By default, this will check out the newest release of `franka_ros`. If you want to build a particular version of `franka_ros` instead, check out the corresponding Git tag:

```
git checkout <version>
```

Install any missing dependencies and build the packages:

```
rosdep install --from-paths src --ignore-src --rosdistro noetic -y --skip-keys
libfranka
catkin_make -DCMAKE_BUILD_TYPE=Release -DFranka_DIR:PATH=/path/to/libfranka/build
source devel/setup.sh
```

Warning

If you also installed `ros-noetic-libfranka`, `libfranka` might be picked up from `/opt/ros/noetic` instead of from your custom `libfranka` build!

Setting up the real-time kernel

In order to control your robot using `libfranka`, the controller program on the workstation PC must run with *real-time priority* under a `PREEMPT_RT` kernel. This section describes the procedure of patching a kernel to support `PREEMPT_RT` and creating an installation package.

Note

NVIDIA drivers are not officially supported on `PREEMPT_RT` kernels, but the installation might work anyway by passing the environment variable `IGNORE_PREEMPT_RT_PRESENCE=1` to the installation command.

First, install the necessary dependencies:

```
sudo apt-get install build-essential bc curl debhelper dpkg-dev devscripts fakeroot
libssl-dev libelf-dev bison flex cpio kmod rsync libncurses-dev
```

Then, you have to decide which kernel version to use. To find the one you are using currently, use `uname -r`. Real-time patches are only available for select kernel versions, see https://www.kernel.org/pub/linux/kernel/projects/rt/. We recommend choosing the version closest to the one you currently use. If you choose a different version, simply substitute the numbers. Having decided on a version, use `curl` to download the source files:

Note

For Ubuntu 16.04 tested with the kernel version 4.14.12:

```
curl -LO https://www.kernel.org/pub/linux/kernel/v4.x/linux-4.14.12.tar.xz
curl -LO https://www.kernel.org/pub/linux/kernel/v4.x/linux-4.14.12.tar.sign
curl -LO https://www.kernel.org/pub/linux/kernel/projects/rt/4.14/older/patch-
4.14.12-rt10.patch.xz
curl -LO https://www.kernel.org/pub/linux/kernel/projects/rt/4.14/older/patch-
4.14.12-rt10.patch.sign
```

Note

For Ubuntu 18.04 tested with the kernel version 5.4.19:

```
curl -LO https://www.kernel.org/pub/linux/kernel/v5.x/linux-5.4.19.tar.xz
curl -LO https://www.kernel.org/pub/linux/kernel/v5.x/linux-5.4.19.tar.sign
curl -LO https://www.kernel.org/pub/linux/kernel/projects/rt/5.4/older/patch-
5.4.19-rt10.patch.xz
curl -LO https://www.kernel.org/pub/linux/kernel/projects/rt/5.4/older/patch-
5.4.19-rt10.patch.sign
```

Note

For Ubuntu 20.04 tested with the kernel version 5.9.1:

```
curl -LO https://www.kernel.org/pub/linux/kernel/v5.x/linux-5.9.1.tar.xz
curl -LO https://www.kernel.org/pub/linux/kernel/v5.x/linux-5.9.1.tar.sign
curl -LO https://www.kernel.org/pub/linux/kernel/projects/rt/5.9/patch-5.9.1-
rt20.patch.xz
curl -LO https://www.kernel.org/pub/linux/kernel/projects/rt/5.9/patch-5.9.1-
rt20.patch.sign
```

Note

For Ubuntu 22.04, we recommend using the Ubuntu Pro real-time kernel. After enabling it, you can skip directly to Allow a user to set real-time permissions for its processes. In case you do not want to use Ubuntu Pro, you can proceed as for the other versions (tested with the kernel version 6.8.0):

```
curl -LO https://www.kernel.org/pub/linux/kernel/v6.x/linux-6.8.tar.xz
curl -LO https://www.kernel.org/pub/linux/kernel/v6.x/linux-6.8.tar.sign
curl -LO https://www.kernel.org/pub/linux/kernel/projects/rt/6.8/older/patch-6.8-
rt8.patch.xz
curl -LO https://www.kernel.org/pub/linux/kernel/projects/rt/6.8/older/patch-6.8-
rt8.patch.sign
```

And decompress them with:

```
xz -d *.xz
```

Verifying file integrity

Note

This step is optional but recommended!

The `.sign` files can be used to verify that the downloaded files were not corrupted or tampered with. The steps shown here are adapted from the Linux Kernel Archive , see the linked page for more details about the process.

You can use `gpg2` to verify the `.tar` archives:

```
gpg2 --verify linux-*.tar.sign
gpg2 --verify patch-*.patch.sign
```

If your output is similar to the following:

```
$ gpg2 --verify linux-*.tar.sign
gpg: assuming signed data in 'linux-4.14.12.tar'
gpg: Signature made Fr 05 Jan 2018 06:49:11 PST using RSA key ID 6092693E
gpg: Can't check signature: No public key
```

You have to first download the public key of the person who signed the above file. As you can see from the above output, it has the ID `6092693E`. You can obtain it from the key server:

```
gpg2  --keyserver hkp://keyserver.ubuntu.com:80 --recv-keys 6092693E
```

Similarly for the patch:

```
gpg2 --keyserver hkp://keyserver.ubuntu.com:80 --recv-keys 2872E4CC
```

Note that keys for other kernel version might have different IDs, you will have to adapt accordingly.

Having downloaded the keys, you can now verify the sources. Here is an example of a correct output:

```
$ gpg2 --verify linux-*.tar.sign
gpg: assuming signed data in 'linux-4.14.12.tar'
gpg: Signature made Fr 05 Jan 2018 06:49:11 PST using RSA key ID 6092693E
gpg: Good signature from "Greg Kroah-Hartman <gregkh@linuxfoundation.org>" [unknown]
gpg:                 aka "Greg Kroah-Hartman <gregkh@kernel.org>" [unknown]
gpg:                 aka "Greg Kroah-Hartman (Linux kernel stable release signing key)
<greg@kroah.com>" [unknown]
gpg: WARNING: This key is not certified with a trusted signature!
gpg:          There is no indication that the signature belongs to the owner.
Primary key fingerprint: 647F 2865 4894 E3BD 4571  99BE 38DB BDC8 6092 693E
```

See Linux Kernel Archive for more information about the warning.

Compiling the kernel

Once you are sure the files were downloaded properly, you can extract the source code and apply the patch:

```
tar xf linux-*.tar
cd linux-*/
patch -p1 < ../patch-*.patch
```

Next copy your currently booted kernel configuration as the default config for the new real time kernel:

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

Exclude the debug information from the kernel files to save space:

```
scripts/config --disable DEBUG_INFO
scripts/config --disable DEBUG_INFO_DWARF_TOOLCHAIN_DEFAULT
scripts/config --disable DEBUG_KERNEL
```

Disable the system-wide ring of trusted keys:

```
scripts/config --disable SYSTEM_TRUSTED_KEYS
scripts/config --disable SYSTEM_REVOCATION_LIST
```

Activate the Fully Preemptible Kernel (Real-Time):

```
scripts/config --disable PREEMPT_NONE
scripts/config --disable PREEMPT_VOLUNTARY
scripts/config --disable PREEMPT
scripts/config --enable PREEMPT_RT
```

Now you can use this config as the default to configure the build:

```
make olddefconfig
```

Afterwards, you are ready to compile the kernel. As this is a lengthy process, set the multithreading option `-j` to the number of your CPU cores:

```
make -j$(nproc) deb-pkg
```

Finally, you are ready to install the newly created package. The exact names depend on your environment, but you are looking for `headers` and `images` packages without the `dbg` suffix. To install:

```
sudo IGNORE_PREEMPT_RT_PRESENCE=1 dpkg -i ../linux-headers-*.deb ../linux-image-*.deb
```

Verifying the new kernel

Restart your system. The Grub boot menu should now allow you to choose your newly installed kernel. To see which one is currently being used, see the output of the `uname -a` command. It should contain the string `PREEMPT RT` and the version number you chose. Additionally, `/sys/kernel/realtime` should exist and contain the the number `1`.

Note

If you encounter errors that you fail to boot the new kernel see Cannot boot realtime kernel because of “Invalid Signature”

Allow a user to set real-time permissions for its processes

After the `PREEMPT_RT` kernel is installed and running, add a group named *realtime* and add the user controlling your robot to this group:

```
sudo addgroup realtime
sudo usermod -a -G realtime $(whoami)
```

Afterwards, add the following limits to the *realtime* group in `/etc/security/limits.conf`:

```
@realtime soft rtprio 99
@realtime soft priority 99
@realtime soft memlock 102400
@realtime hard rtprio 99
@realtime hard priority 99
@realtime hard memlock 102400
```

The limits will be applied after you log out and in again.