Make a ROS Launch Start on Boot (robot_upstart)

So, you've developed a nice ROS application, which can be launched with a single launch file: great!

But… There is still a problem. Actually you have to manually launch this file with a command line tool (roslaunch), and you wonder how you could make your application automatically start when you boot your computer… Or your Raspberry Pi!

It could be great if you could just power on your laptop/Raspberry Pi, and after a few seconds, your robot automatically starts without you having to ssh inside and run the command. So you can have a fully autonomous embedded computer in your robot.

Well, there is a solution, called robot_upstart.

In this tutorial I'll show you how to install the robot_upstart package, and how to use it so you won't have to manually launch your application anymore. Also I'll give you some useful tips so you save time on potential future headaches.

This tutorial works on all computers using Ubuntu, and especially Raspberry Pi 4 (and earlier 3B, 3B+) boards. If you're using a Pi, make sure you have correctly installed ROS on it.

Table of Contents

Setup for robot_upstart

Install robot_upstart package

To install the package, simply run `sudo apt-get install ros-<distro>-robot-upstart`, and replace "<distro>" by your ROS version: kinetic, melodic, …

That's all you need to do.

Basic node and launch file for this example

For this example on how to use robot_upstart, we'll use one node and one launch file.

This is the node, located in the 'my_robot_tutorials' package for this tutorial.


```python
#!/usr/bin/env python

import rospy
from std_msgs.msg import Int64

if __name__ == "__main__":
    rospy.init_node("counter_publisher")

    rate = rospy.get_param("/counter_publisher_rate")
    counter = 0
    pub = rospy.Publisher("counter", Int64, queue_size=1)
    rate = rospy.Rate(rate)
    rospy.loginfo("Starting publishing...")

    while not rospy.is_shutdown():
        pub.publish(counter)
        counter += 1
        rate.sleep()
```

And this is the launch file, located in the 'my_robot_bringup' package, inside the 'launch/' folder.

```
<launch>
    <param name="/counter_publisher_rate" type="int" value="5" />
    <node name="counter_publisher" pkg="my_robot_tutorials"
type="counter.py" output="screen"/>
</launch>
```

When you launch this file with `roslaunch my_robot_bringup my_robot.launch`, the node is started, gets one parameter, and starts to publish a counter at the frequency given by the parameter, using a ROS Rate.

Now that we have everything setup, let's make this ROS launch file start on boot!

robot_upstart install script

We'll use the "install" script from the robot_upstart package to make a launch file start on boot.

Here you have 2 options: using only the terminal, or the robot_upstart Python API.

The install script has many possible arguments, here we'll only see the basic ones so you can quickly get started.

Install script from terminal

You can run the script via rosrun. The package name is "robot_upstart" and the script name is "install". Note that usually, when you use rosrun you have to start a ROS master before, or else you'll get a "Unable to

register with master node" error. Here, no need to do that, the script won't create a node when executed.

Execute this command:

```
rosrun robot_upstart install my_robot_bringup/launch/my_robot.launch
--job my_robot_ros --symlink
```

Here's the outcome when you run the robot_upstart install script:


```
$ rosrun robot_upstart install my_robot_bringup/launch/my_robot.launch
--job my_robot_ros
/lib/systemd/systemd
Preparing to install files to the following paths:
  /etc/ros/melodic/my_robot_ros.d/.installed_files
  /etc/ros/melodic/my_robot_ros.d/my_robot.launch
  /etc/systemd/system/multi-user.target.wants/my_robot_ros.service
  /lib/systemd/system/my_robot_ros.service
  /usr/sbin/my_robot_ros-start
  /usr/sbin/my_robot_ros-stop
Now calling: /usr/bin/sudo /opt/ros/melodic/lib/robot_upstart
/mutate_files
[sudo] password for user:
Filesystem operation succeeded.
** To complete installation please run the following command:
 sudo systemctl daemon-reload && sudo systemctl start my_robot_ros
```

You will have to give your user's password to complete the installation. After that, just run `sudo systemctl daemon-reload`. The launch file is now correctly installed, but is not running yet.

As you can see, the script created a bunch of files in different places. Behind the hood, robot_upstart uses systemd to create executables that run on boot. For more info on how to use systemd on your Raspberry Pi (for any executable, not ROS specific), check out this tutorial.

So, in this case there is one executable to start the launch file, and one to stop it. On boot, systemd will call the first one automatically, and your application will start in the background.

Here's a breakdown of all the arguments we used:

- `my_robot_bringup/launch/my_robot.launch`: the first argument is

the relative path to the launch file, from the catkin workspace src/ folder.

- `--job my_robot_ros` : you can choose to give a name to the "Job" that will be created. This name will be used to create the systemd job.

- `--symlink` : By default, the install script will copy the content of your launch file and run this copy. Thus, every time you update your launch file you have to remember to also update the copy. Here, problem solved: the install script will create a symbolic link to your original launch file.

Testing time

You can test if the installation went correctly, without having to reboot your computer/Raspberry Pi.

Just run those 2 commands to start and stop your ROS launch file:

```
$ sudo systemctl start my_robot_ros.service
$ rosnode list
/counter_publisher
/rosout

...

$ sudo systemctl stop my_robot_ros.service
$ rosnode list
ERROR: Unable to communicate with master!
```

This is very handy when you want to debug your programs, so you don't need to reboot.

And as you can see, nothing is displayed on the screen when you start the application. This is because it's running in the background.

Now, you can reboot your computer/Raspberry Pi, and check that the program has started on boot. You should get the counter node with `rosnode list`, and be able to see the values published on the /counter topic with `rostopic echo /counter`. You don't need to manually start the launch file anymore. Success!

One more thing: you can also try to change the "/counter_publisher_rate" parameter in "my_robot.launch". When you restart "my_robot_ros.service", the parameter's value will be changed. But if you didn't use the –symlink option with the install script, the new value wouldn't have been used.

```
$ sudo systemctl start my_robot_ros.service
$ rosparam get /counter_publisher_rate
5

...
# Modify your launch file and change the parameter's value
...

$ sudo systemctl start my_robot_ros.service
$ rosparam get /counter_publisher_rate
1
```

robot_upstart install with Python API

Let's see how you can use the robot_upstart Python API to install your launch file. Note that it's the exact same thing as running the script in the terminal.

Using the Python API may be more readable if you happen to have a lot of arguments, and you can also use this script inside other Python code.

```
1. import robot_upstart
2.
3. j = robot_upstart.Job(name="my_robot_ros")
4. j.symlink = True
5.
6. j.add(package="my_robot_bringup",
   filename="launch/my_robot.launch")
7.
8. j.install()
```


First you import the robot_upstart package. Then you can create a "Job" object and give it a name. There is no argument to set the

symlink option here, so you have to directly set the symlink attribute to True (default is False).

Then, you can use the method add() to add a launch file, by giving the package name and the file name inside this package – notice the relative path here, from the package's name and not from the catkin workspace src/ folder.

Finally, the install() method will execute the script as you previously did on the terminal.

Usually, this Python file is place inside a scripts/ folder in the my_robot_bringup package. You can follow this "convention" if you want.

Uninstall

Let's now see how to uninstall a job that you created with the robot_upstart install script.

Disabling

Before uninstalling the full thing, you have to know that you can simply disable the execution of the launch file on boot, without having to uninstall it.

As it's running with systemd, you can simply run:

```
$ sudo systemctl disable my_robot_ros.service
Removed /etc/systemd/system/multi-user.target.wants
/my_robot_ros.service.
```

This will keep the files created before and the executables, but it will prevent systemd from starting the launch file on boot. You can still use `sudo systemctl start my_robot_ros.service` and `sudo systemctl stop my_robot_ros.service` to start and stop the launch file from the terminal.

To re-enable the systemd job, run:

```
$ sudo systemctl enable my_robot_ros.service
Created symlink /etc/systemd/system/multi-user.target.wants
/my_robot_ros.service → /lib/systemd/system/my_robot_ros.service.
```

Uninstall from terminal

Now, if you really want to completely uninstall, run `rosrun robot_upstart uninstall my_robot_ros`, where "my_robot_ros" is the name of the job you used with the install script.

Here's the result you should see:

```
$ rosrun robot_upstart uninstall my_robot_ros
/lib/systemd/systemd
Preparing to remove the following paths:
  /etc/ros/melodic/my_robot_ros.d
  /etc/ros/melodic/my_robot_ros.d/.installed_files
  /etc/systemd/system/multi-user.target.wants/my_robot_ros.service
  /lib/systemd/system/my_robot_ros.service
  /usr/sbin/my_robot_ros-start
  /usr/sbin/my_robot_ros-stop
Now calling: /usr/bin/sudo /opt/ros/melodic/lib/robot_upstart
/mutate_files
Filesystem operation succeeded.
```

Basically this will remove all files created when you ran the install script.

robot_upstart uninstall with Python API

As we did for the robot_upstart install script, let's just write a Python code that will do the exact same thing as using the uninstall script in the terminal.

```python
import robot_upstart

j = robot_upstart.Job(name="my_robot_ros")

j.uninstall()
```


You just need to create a "Job" object and give the name of the job you previously created when installing. Then, calling the uninstall() method will simply execute the uninstall script.

Well, that's about it for installing and uninstalling a job to start a ROS launch file on boot. Quite simple, isn't it?

Warning about hardware permissions

Please read – it will save you a lot of time!

Let's say your computer/Raspberry Pi is talking to another device via USB/Serial communication – which is not an uncommon thing to do in a robotics application. In this case, you have to make your user belong to the "dialout" group.

Now, when you install a ROS launch file with robot_upstart, things are

different when it comes to hardware permissions.

When the launch file is started on boot with the job you created in the install script, the user will be the same user who installed the script, but it will be an **unprivileged user**. You loose the group memberships you've setup before.

As a result, your program may crash because you don't have the necessary hardware permissions anymore!

Fortunately there is a quick way to solve that, by creating a udev rule.

Create a new file named "local.rules" in /etc/udev/rules.d/.

```
cd /etc/udev/rules.d/
sudo touch local.rules.d
```

In this file (edit with admin rights), add a new line: `ACTION=="add", KERNEL=="dialout", MODE="0666"`. This will make the "dialout" group available for all unprivileged users.

You need to reboot so the new settings can apply.

For more info check out: Hardware permissions with udev rules.

Going further

In this tutorial you have discovered how to make your ROS launch file start on boot, thanks to the robot_upstart package. You know how to install and uninstall a startup job, both from the terminal or using the Python API. Also, you know how to do basic debugging on those startup jobs you have created.

To go further, check out those resources:

- Read the robot_upstart install script documentation: some additional arguments may be useful, especially if you have a multi-machine setup.
- Learn more about systemd, so you can better understand what's going on under the hood.

Raspberry Pi Tutorials, ROS Tutorials

Did you find this tutorial useful?

Do you want to learn how to program with ROS?

If yes, this course is for you:

ROS For Beginners - A Step By Step Course

>> ROS For Beginners - A Step By Step Course <<

Leave a Comment

Name *

Email *

Website

Post Comment