

Note: This tutorial assumes that you have completed the previous tutorials: building a ROS package (/ROS/Tutorials/BuildingPackages).

💡 Please ask about problems and questions regarding this tutorial on answers.ros.org (<http://answers.ros.org>). Don't forget to include in your question the link to this page, the versions of your OS & ROS, and also add appropriate tags.

Understanding ROS Nodes

Description: This tutorial introduces ROS graph concepts and discusses the use of roscore (/roscore), rosnode (/rosnode), and rosrund (/rosrund) commandline tools.

Tutorial Level: BEGINNER

Next Tutorial: Understanding ROS topics (/ROS/Tutorials/UnderstandingTopics)

Conteúdo

1. Prerequisites
2. Quick Overview of Graph Concepts
3. Nodes
4. Client Libraries
5. roscore
6. Using rosnode
7. Using rosrund
8. Review
9. Video Demonstration

1. Prerequisites

For this tutorial we'll use a lightweight simulator, to install it run the following command:

```
$ sudo apt-get install ros-<distro>-ros-tutorials
```

Replace '<distro>' with the name of your ROS distribution (e.g. indigo, jade, kinetic, noetic)

2. Quick Overview of Graph Concepts

- Nodes (/Nodes): A node is an executable that uses ROS to communicate with other nodes.
- Messages (/Messages): ROS data type used when subscribing or publishing to a topic.
- Topics (/Topics): Nodes can *publish* messages to a topic as well as *subscribe* to a topic to receive messages.
- Master (/Master): Name service for ROS (i.e. helps nodes find each other)
- rosout (/rosout): ROS equivalent of stdout/stderr
- roscore (/roscore): Master + rosout + parameter server (parameter server will be introduced later)

3. Nodes

A node really isn't much more than an executable file within a ROS package. ROS nodes use a ROS client library to communicate with other nodes. Nodes can publish or subscribe to a Topic. Nodes can also provide or use a Service.

4. Client Libraries

ROS client libraries allow nodes written in different programming languages to communicate:

- rospy = python client library
- roscpp = c++ client library

5. roscore

roscore is the first thing you should run when using ROS.

Please run:

```
$ roscore
```

You will see something similar to:

```
... logging to ~/.ros/log/9cf88ce4-b14d-11df-8a75-00251148e8cf/roslaunch-
machine_name-13039.log
Checking log directory for disk usage. This may take awhile.
Press Ctrl-C to interrupt
Done checking log file disk usage. Usage is <1GB.

started roslaunch server http://machine_name:33919/
ros_comm version 1.4.7

SUMMARY
=====

PARAMETERS
* /rosversion
* /rostdistro

NODES

auto-starting new master
process[master]: started with pid [13054]
ROS_MASTER_URI=http://machine_name:11311/

setting /run_id to 9cf88ce4-b14d-11df-8a75-00251148e8cf
process[rosout-1]: started with pid [13067]
started core service [/rosout]
```

If roscore does not initialize, you probably have a network configuration issue. See [Network Setup](#) -

Single Machine Configuration (http://www.ros.org/wiki/ROS/NetworkSetup#Single_machine_configuration)

If `roscore` does not initialize and sends a message about lack of permissions, probably the `~/ . ros` folder is owned by `root`, change recursively the ownership of that folder with:

```
$ sudo chown -R <your_username> ~/ . ros
```

6. Using rosnode

Open up a **new terminal**, and let's use **roscnode** to see what running `roscore` did... Bear in mind to keep the previous terminal open either by opening a new tab or simply minimizing it.

Note: When opening a new terminal your environment is reset and your `~/ . bashrc` file is sourced. If you have trouble running commands like `roscnode` then you might need to add some environment setup files to your `~/ . bashrc` or manually re-source them.

`roscnode` displays information about the ROS nodes that are currently running. The `roscnode list` command lists these active nodes:

```
$ roscnode list
```

You will see:

```
/rosout
```

This showed us that there is only one node running: `rosout (/rosout)`. This is always running as it collects and logs nodes' debugging output.

The `roscnode info` command returns information about a specific node.

```
$ roscnode info /rosout
```

This gave us some more information about `rosout`, such as the fact that it publishes `/rosout_agg`.

```
-----
Node [/rosout]
Publications:
  * /rosout_agg [rosgraph_msgs/Log]

Subscriptions:
  * /rosout [unknown type]

Services:
  * /rosout/get_loggers
  * /rosout/set_logger_level

contacting node http://machine_name:54614/ ...
Pid: 5092
```

Now, let's see some more nodes. For this, we're going to use `ros run` to bring up another node.

7. Using `roslaunch`

`roslaunch` allows you to use the package name to directly run a node within a package (without having to know the package path).

Usage:

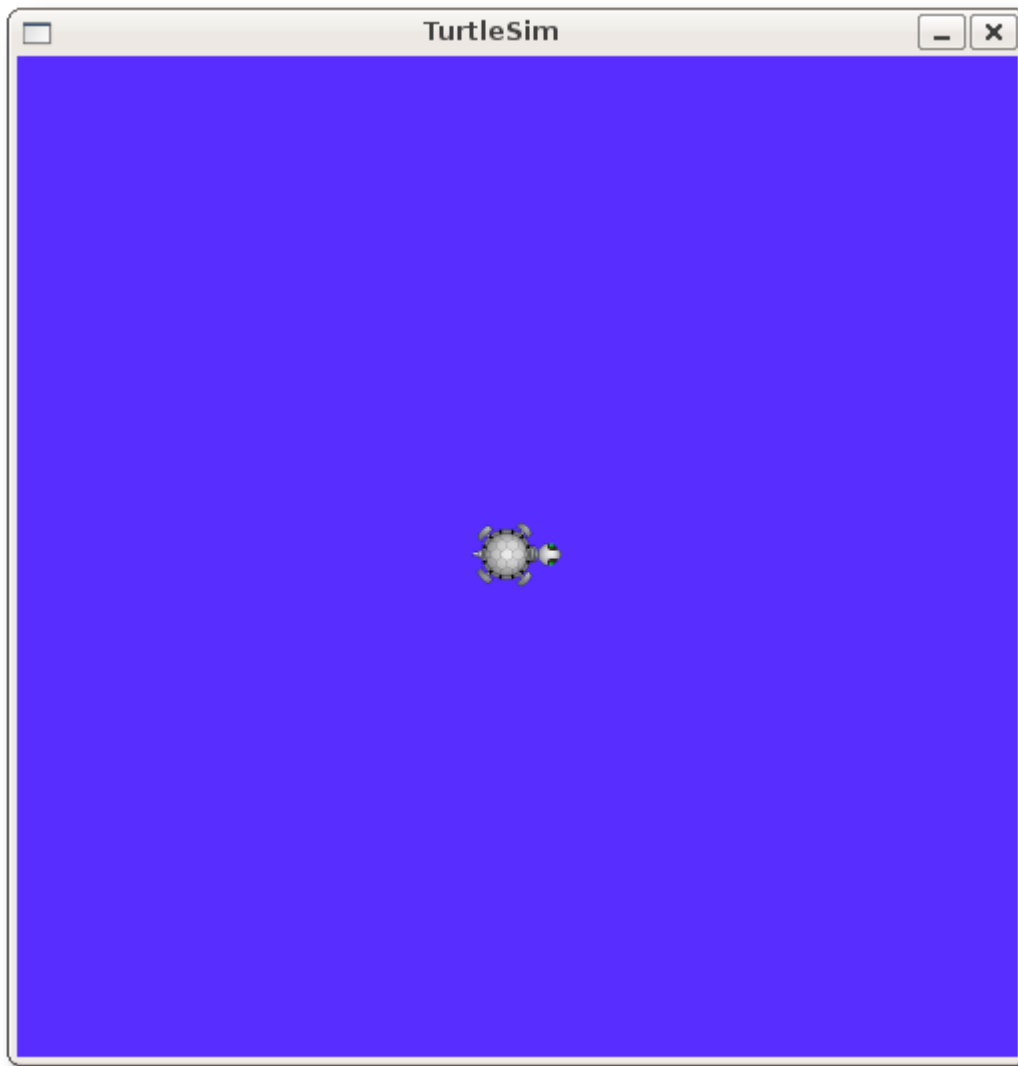
```
$ roslaunch [package_name] [node_name]
```

So now we can run the `turtlesim_node` in the `turtlesim` package.

Then, in a **new terminal**:

```
$ roslaunch turtlesim turtlesim_node
```

You will see the `turtlesim` window:



NOTE: The turtle may look different in your `turtlesim` window. Don't worry about it - there are many types of turtle ([/Distributions#Current_Distribution_Releases](#)) and yours is a surprise!

In a **new terminal**:

```
$ rosnode list
```

You will see something similar to:

```
/rosout  
/turtlesim
```

One powerful feature of ROS is that you can reassign Names from the command-line.

Close the turtlesim window to stop the node (or go back to the `roslaunch turtlesim turtlesim.launch` terminal and use `ctrl-C`). Now let's re-run it, but this time use a Remapping Argument (`/Remapping%20Arguments`) to change the node's name:

```
$ roslaunch turtlesim turtlesim.launch __name:=my_turtle
```

Now, if we go back and use `roslaunch turtlesim turtlesim.launch`:

```
$ roslaunch turtlesim turtlesim.launch
```

You will see something similar to:

```
/my_turtle  
/rosout
```

Note: If you still see `/turtlesim` in the list, it might mean that you stopped the node in the terminal using `ctrl-C` instead of closing the window, or that you don't have the `$ROS_HOSTNAME` environment variable defined as described in [Network Setup - Single Machine Configuration](http://www.ros.org/wiki/ROS/NetworkSetup#Single_machine_configuration) (http://www.ros.org/wiki/ROS/NetworkSetup#Single_machine_configuration). You can try cleaning the `roslaunch` list with: `roslaunch clean`

We see our new `/my_turtle` node. Let's use another `roslaunch` command, `ping`, to test that it's up:

```
$ roslaunch turtlesim turtlesim.launch
```

```
roslaunch: node is [/my_turtle]  
pinging /my_turtle with a timeout of 3.0s  
xmlrpc reply from http://aqy:42235/      time=1.152992ms  
xmlrpc reply from http://aqy:42235/      time=1.120090ms  
xmlrpc reply from http://aqy:42235/      time=1.700878ms  
xmlrpc reply from http://aqy:42235/      time=1.127958ms
```

8. Review

What was covered:

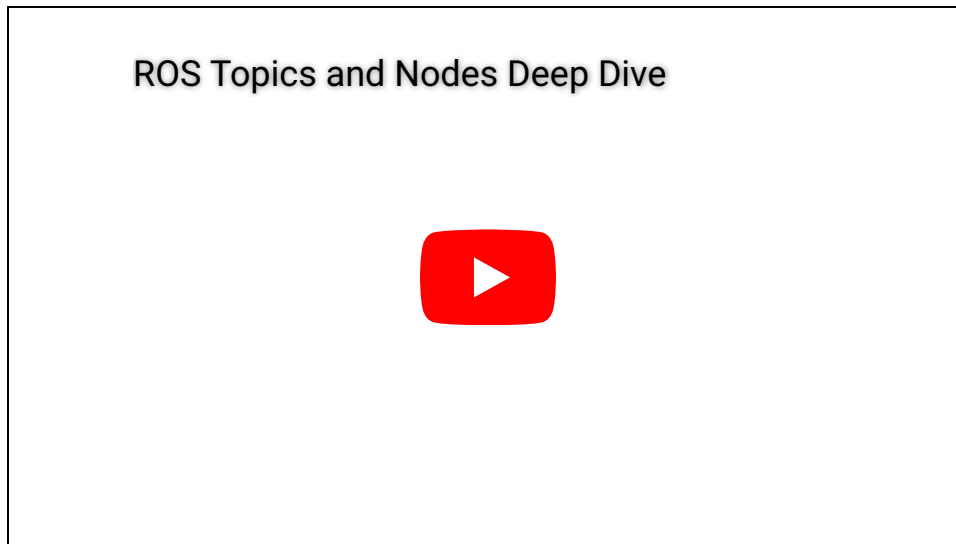
- `roscore` = `ros+core` : master (provides name service for ROS) + `rosout` (stdout/stderr) + parameter server (parameter server will be introduced later)
- `roslaunch` = `ros+launch` : ROS tool to get information about a node.

- `roslaunch` = `ros+launch` : runs a node from a given package.

Now that you understand how ROS nodes work, let's look at how ROS topics work (/ROS/Tutorials/UnderstandingTopics). Also, feel free to press `Ctrl - C` to stop `turtlesim_node`.

9. Video Demonstration

Watch the video below to have more explanation on Python Nodes Communication and step by step guide .



Wiki: ROS/Tutorials/UnderstandingNodes (última edição 2022-10-18 16:16:55 efectuada por Muhammad Luqman (/Muhammad%20Luqman))

Except where otherwise noted, the ROS wiki is licensed under the Creative Commons Attribution 3.0 (<http://creativecommons.org/licenses/by/3.0/>)

Brought to you by:  Open Robotics

(<https://www.openrobotics.org/>)