**Nota:** Este tutorial pressupõe que você concluiu os tutoriais anteriores: construindo um pacote ROS (/ROS/Tutorials/BuildingPackages) .

Por favor, pergunte sobre problemas e perguntas relacionadas a este tutorial em Answers.ros.org (http://answers.ros.org) . Não se esqueça de incluir na sua pergunta o link para esta página, as versões do seu SO e ROS, e também adicionar as tags apropriadas.

# Compreendendo os nós ROS

**Descrição:** Este tutorial apresenta conceitos de gráfico ROS e discute o uso de ferramentas de linha de comando roscore (/roscore) , rosnode (/rosnode) e rosrun (/rosrun) .

**Nível do Tutorial:** INICIANTE

**Próximo Tutorial:** Compreendendo os tópicos do ROS (/ROS/Tutorials/UnderstandingTopics)

**Conteúdo**



# 1. Pré-requisitos

Para este tutorial usaremos um simulador leve, para instalá-lo execute o seguinte comando:

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

Substitua '<distro>' pelo nome da sua distribuição ROS (por exemplo, índigo, jade, cinético, noético)

# 2. Visão geral rápida dos conceitos de gráfico

- Nós (/Nodes) : um nó é um executável que usa ROS para se comunicar com outros nós.
- Mensagens (/Messages) : tipo de dados ROS usado ao assinar ou publicar um tópico.
- Tópicos (/Topics) : os nós podem *publicar* mensagens em um tópico, bem como *assinar* um tópico para receber mensagens.
- Mestre (/Master) : serviço de nomes para ROS (ou seja, ajuda os nós a se encontrarem)
- rosout (/rosout) : equivalente ROS de stdout/stderr
- roscore (/roscore) : Master + rosout + servidor de parâmetros (o servidor de parâmetros será

apresentado posteriormente)

# 3. Nós

Na verdade, um nó não é muito mais do que um arquivo executável dentro de um pacote ROS. Os nós ROS usam uma biblioteca cliente ROS para se comunicar com outros nós. Os nós podem publicar ou assinar um tópico. Os nós também podem fornecer ou usar um serviço.

# 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
 * /rosdistro

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 **rosnode** 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 `rosnode` then you might need to add some environment setup files to your `~/.bashrc` or manually re-source them.

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

```
$ rosnode 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 `rosnode info` command returns information about a specific node.

```
$ rosnode 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 `rosrun` to bring up another node.

# 7. Using rosrun

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

Usage:

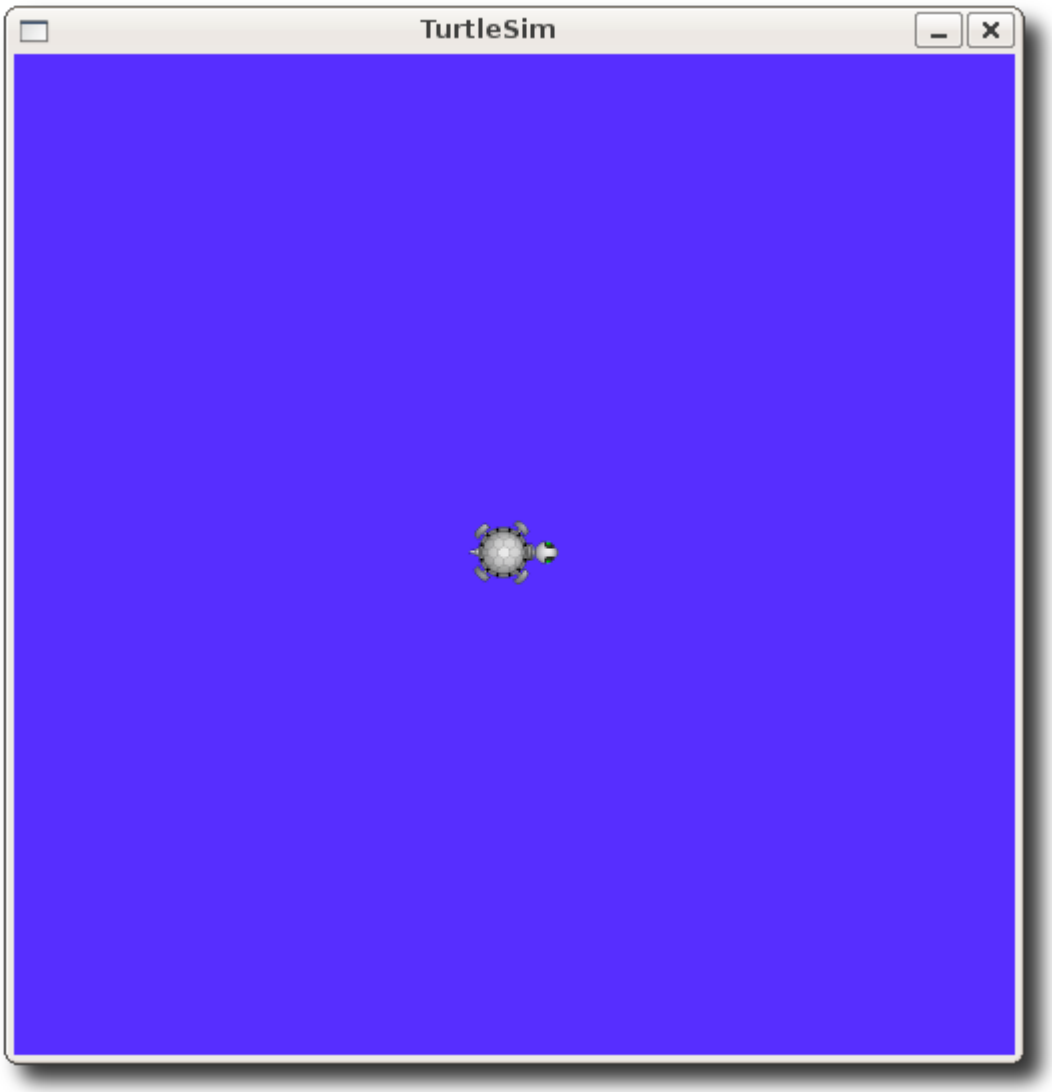
```
$ rosrun [package_name] [node_name]
```

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

Then, in a **new terminal**:

```
$ rosrun 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 `rosrun turtlesim` 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:

```
$ rosrun turtlesim turtlesim_node __name:=my_turtle
```

Now, if we go back and use `rosnode list`:

```
$ rosnode list
```

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). You can try cleaning the rosnode list with: $ rosnode cleanup

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

```
$ rosnode ping my_turtle
```

```
rosnode: 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)
- rosnode = ros+node : ROS tool to get information about a node.
- rosrun = ros+run : 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))