ROS Cheat sheet -

If you installed ROS Indigo, that would be:

$ source /opt/ros/indigo/setup.bash

You will need to run this command on every new shell you open to have access to the ros commands, unless you add this line to your .bashrc. This process allows you to install several ROS distributions (e.g. fuerte and groovy) on the same computer and switch between them.

After pushing the **TAB** key, the command line should fill out as much as possible:

$ roscd turtle

However, in this case there are multiple packages that begin with turtle. Try typing **TAB** another time. This should display all the ROS packages that begin with turtle:

* rospack = ros + pack(age)
* roscd = ros + cd
* rosls = ros + ls

To add the workspace to your ROS environment you need to source the generated setup file:

$ . ~/catkin\_ws/devel/setup.bash

$ rospack depends1 beginner\_tutorials

* std\_msgs  
  rospy  
  roscpp

As you can see, rospack lists the same dependencies that were used as arguments when running catkin\_create\_pkg. These dependencies for a package are stored in the **package.xml** file:

You should already have a [catkin workspace](http://wiki.ros.org/catkin/Tutorials/create_a_workspace) and a new catkin package called beginner\_tutorials from the previous tutorial, [Creating a Package](http://wiki.ros.org/ROS/Tutorials/CreatingPackage). Go into the catkin workspace if you are not already there and look in the src folder:

$ cd ~/catkin\_ws/  
$ ls src

* beginner\_tutorials/ CMakeLists.txt@

You should see that there is a folder called beginner\_tutorials which you created with [catkin\_create\_pkg](http://wiki.ros.org/catkin/commands/catkin_create_pkg) in the previous tutorial. We can now build that package using [catkin\_make](http://wiki.ros.org/catkin/commands/catkin_make):

$ catkin\_make

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

Please run:

$ roscore

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

$ rosnode list

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

* 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.

rqt\_graph creates a dynamic graph of what's going on in the system. rqt\_graph is part of the rqt package.

**In a new terminal**:

$ rosrun rqt\_graph rqt\_graph

Communication on topics happens by sending ROS **messages** between nodes. For the publisher (turtle\_teleop\_key) and subscriber (turtlesim\_node) to communicate, the publisher and subscriber must send and receive the same **type** of message. This means that a topic **type** is defined by the message **type** published on it. The **type** of the message sent on a topic can be determined using rostopic type.

#### **Using rostopic type**

rostopic type returns the message type of any topic being published.

Usage:

rostopic type [topic]