Introduction to Basic ROS Tools

David Swords

University College Dublin

david.swords@ucdconnect.ie

January 24, 2014

Installation

Ubuntu 12.04LTS

```
sudo sh -c 'echo "deb http://packages.ros.org/ros/ubuntu
 precise main" > /etc/apt/sources.list.d/ros-latest.list'
```

Ubuntu 13.04LTS

```
sudo sh -c 'echo "deb http://packages.ros.org/ros/ubuntu
 raring main" > /etc/apt/sources.list.d/ros-latest.list'
```

Setting Up Keys

```
wget http://packages.ros.org/ros.key -O - | sudo apt-key
 add -
```

Update Package Listings

```
sudo apt-get update
```

Installation

Grabbing ROS Packages

```
sudo apt-get install ros-hydro-desktop
```

Initializing rosdep

```
sudo rosdep init
rosdep update
```

Environmental Setup

```
echo "source /opt/ros/hydro/setup.bash" >> ~/.bashrc
source ~/.bashrc
```

Getting rosinstall

```
sudo apt-get install python-rosinstall
```

Creating a ROS Workspace

- catkin is the official build system of ROS
- catkin combines CMake macros and Python scripts to provide some functionality on top of CMake's normal workflow

Making Workspace Directory

```
mkdir -p ~/catkin_ws/src
```

Entering catkin Workspace src

```
cd ~/catkin_ws/src
```

Initializing catkin Workspace

```
catkin_init_workspace
```

Creating a ROS Workspace

Test Build catkin Workspace

```
cd ~/catkin_ws/
catkin_make
```

- `catkin_make` is a convenience tool for working with catkin workspaces
- current directory should now have 'build' and 'devel' folders.
- 'devel' folder now has several setup.*sh files.
- Sourcing will overlay this workspace on top of your environment.

Add Workspace to `ROS_PACKAGE_PATH`

```
source devel/setup.bash
```

Checking `ROS_PACKAGE_PATH`

```
echo $ROS_PACKAGE_PATH
```

rospack

- rospack allows you to get information about packages

Usage

```
rospack find [package_name]
```

Try

```
rospack find roscpp
```

Example (Would return)

```
/opt/ros/hydro/share/roscpp
```

roscd

- roscd is part of the rosbash suite.
- allows you to 'cd' directly to a package or a stack

Usage

```
roscd [locationname[/subdir]]
```

Try

```
roscd roscpp
```

Check

```
pwd
```

Example (Would return)

```
/opt/ros/hydro/share/roscpp
```

rosls

- rosls is part of the rosbash suite. It allows you to 'ls' directly in a package by name rather than by absolute path

Usage

```
rosls [locationname[/subdir]]
```

Try

```
rosls roscpp_tutorials
```

Example (Would return)

```
roscppConfig.cmake roscppConfig-version.cmake
roscpp-msg-extras.cmake roscpp-msg-paths.cmake
```

Tab Completion with `roscd`

- tedious to type out an entire package name

Usage

```
roscd roscpp_tut<<< now push the TAB key >>>
```

Example (Would return)

```
roscd roscpp_tutorials/
```

Now Try

```
roscd tur<<< now push the TAB key >>>
```

Example (Would return)

```
roscd turtle
```

Tab Completion with `roscd`

- Though there are multiple packages starting with turtle

Example (Double Tab Results)

```
turtle_actionlib/  turtlesim/   turtle_tf/
```

Try

```
roscd turtles<<< now push the TAB key >>>
```

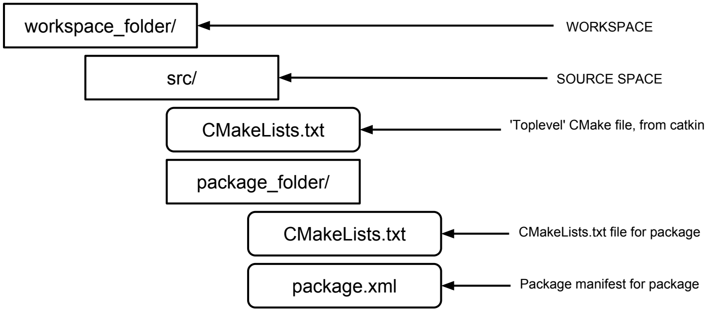
To Finish

```
roscd turtlesim/
```

- More from turtlesim later...

Creating ROS Package

- What makes up a catkin package?
  - must contain a catkin compliant package.xml file, which provides meta information about the package
  - must contain a CMakeLists.txt which uses catkin and metapackages must have a boilerplate CMakeLists.txt file
  - can be no more than one package in each folder, which means no nested packages nor multiple packages sharing the same directory



Creating a catkin Package

First, go back to your workspace

```
cd ~/catkin_ws/src
```

- use the catkin_create_pkg script to create a new package called beginner_tutorials which depends on std_msgs, roscpp, and rospy

Usage

```
catkin_create_pkg <package_name> [depend1] [depend2]
```

Try

```
catkin_create_pkg beginner_tutorials std_msgs rospy roscpp
```

Creating a catkin Package

First, go back to your workspace

```
cd ~/catkin_ws/src
```

- use the catkin_create_pkg script to create a new package called beginner_tutorials which depends on std_msgs, roscpp, and rospy

Usage

```
catkin_create_pkg <package_name> [depend1] [depend2]
```

Try

```
catkin_create_pkg beginner_tutorials std_msgs rospy roscpp
```

Building a catkin Package

- catkin_make adds some convenience to the standard catkin workflow.
- catkin_make combines the calls to cmake and make in the standard CMake workflow.

Example (Typical CMake Workflow)

```
# In a CMake project
$ mkdir build
$ cd build
$ cmake ..
$ make
$ make install  # (optionally)
```

Example (catkin Workflow)

```
# In a catkin workspace
$ catkin_make
$ catkin_make install  # (optionally)
```

Building a catkin Package

Go to the catkin workspace

```
cd ~/catkin_ws/
ls src
```

Example (Would return)

```
beginner_tutorials/
```

Build beginner_tutorials with catkin_make

```
catkin_make
```

- The above commands will build any catkin projects found in the src folder
- 'build' folder is the default location of the build space and is where cmake and make are called to configure and build your packages

Understanding ROS Nodes

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

roscore

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

Try

```
roscore
```

Example (Would return)

```
started roslaunch server http://mbp:56812/
ros_comm version 1.9.53


SUMMARY
========

PARAMETERS
 * /rosdistro
 * /rosversion

NODES

auto-starting new master
process[master]: started with pid [14261]
ROS_MASTER_URI=http://mbp:11311/

setting /run_id to c332b8f6-8478-11e3-8989-f81edfe8843e
process[rosout-1]: started with pid [14275]
started core service [/rosout]
```

rosnode list

- Open up a new terminal, and let's use `rosnode` to see what running `roscore` did...
- `rosnode` displays information about the ROS nodes that are currently running. The `rosnode list` command lists active nodes:

Try (In new terminal)

```
rosnode list
```

Example (Would return)

```
/rosout
```

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

rosnode info

- `rosnode info` command returns information about a specific node did...

Try

```
rosnode info /rosout
```

Example (Would return)

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

Subscriptions:
 * /rosout [unknown type]

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


contacting node http://mbp:49169/ ...
Pid: 14275
```

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]
```

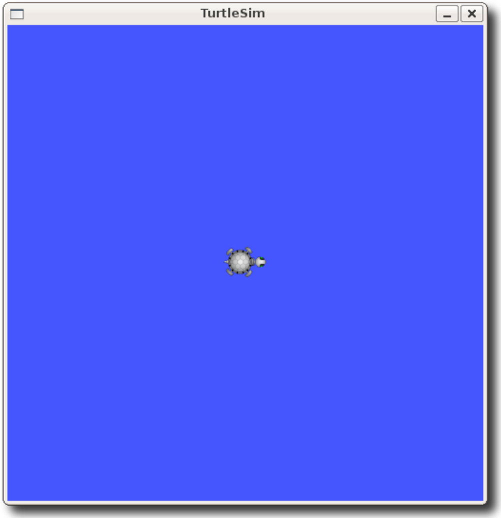
- Now, let's return to the `turtlesim` package and run `turtlesim_node`

Try

```
rosrun turtlesim turtlesim_node
```

- And you should get the following `turtlesim` window...

turtlesim



Remapping with `rosrun`

- Return to the terminal and close the `turtlesim` node with `Ctrl + c`
- A useful feature is the remapping argument with rosrun

Try

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

Try (In another terminal)

```
rosnode list
```

Example (Would return)

```
/rosout
/my_turtle
```

rosrun ping

- You can also ping nodes, useful when testing connectivity over a network

Try (In another terminal)

```
rosnode ping my_turtle
```

Example (Would return)

```
rosnode: node is [/my_turtle]
pinging /my_turtle with a timeout of 3.0s
xmlrpc reply from http://mbp:55488/ time=0.558853ms
xmlrpc reply from http://mbp:55488/ time=1.357794ms
xmlrpc reply from http://mbp:55488/ time=0.741005ms
xmlrpc reply from http://mbp:55488/ time=1.497984ms
xmlrpc reply from http://mbp:55488/ time=0.793934ms
xmlrpc reply from http://mbp:55488/ time=0.932932ms
xmlrpc reply from http://mbp:55488/ time=0.772953ms
xmlrpc reply from http://mbp:55488/ time=0.925064ms
```

Understanding ROS Topics

- Let's go ahead and take control of the `my_turtle` node

Try

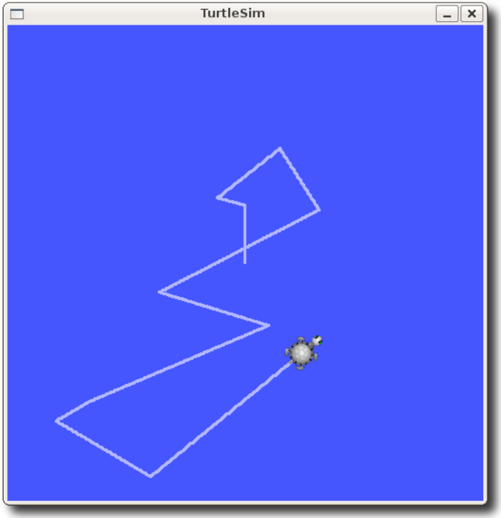
```
rosrun turtlesim turtle_teleop_key
```

Example (Would return)

```
Reading from keyboard
---------------------------
Use arrow keys to move the turtle.
```

- From here we can simply use the directional keys to control the `turtlesim`..

turtlesim



rqt_graph

- The my_turtle node and turtle_teleop_key node are communicating with each other over a ROS Topic
- turtle_teleop_key is publishing the key strokes on a topic, while turtlesim subscribes to the same topic to receive the key strokes
- rqt_graph which shows the nodes and topics currently running

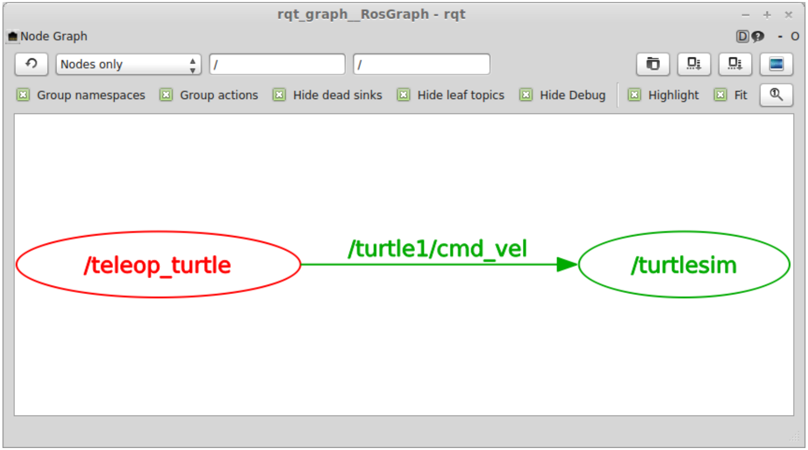
Installing rqt_graph

```
sudo apt-get install ros-hydro-rqt
```

Try

```
rosrun rqt_graph rqt_graph
```

rqt_graph


rqt_graph__RosGraph - rqt
Node Graph
Nodes only
/
/
Group namespaces
Group actions
Hide dead sinks
Hide leaf topics
Hide Debug
Highlight
Fit
/teleop_turtle
/turtle1/cmd_vel
/turtlesim

rostopic

- rostopic allows you to get information about ROS topics
- You can use the help option to get the available sub-commands for rostopic

Try

```
rostopic -h
```

Example (Would return)

```
Commands:
rostopic bw display bandwidth used by topic
rostopic echo print messages to screen
rostopic find find topics by type
rostopic hz display publishing rate of topic
rostopic info print information about active topic
rostopic list list active topics
rostopic pub publish data to topic
rostopic type print topic type
```

rostopic list

- `rostopic list` shows all the currently active topics

Try

```
rostopic list
```

Example (Would return)

```
/rosout
/rosout_agg
/turtle1/cmd_vel
/turtle1/color_sensor
/turtle1/pose
```

rostopic echo

Usage

```
rostopic echo [topic]
```

- Let's look at the data published on the `/turtle1/cmd_vel` topic by the `turtle_teleop_key` node

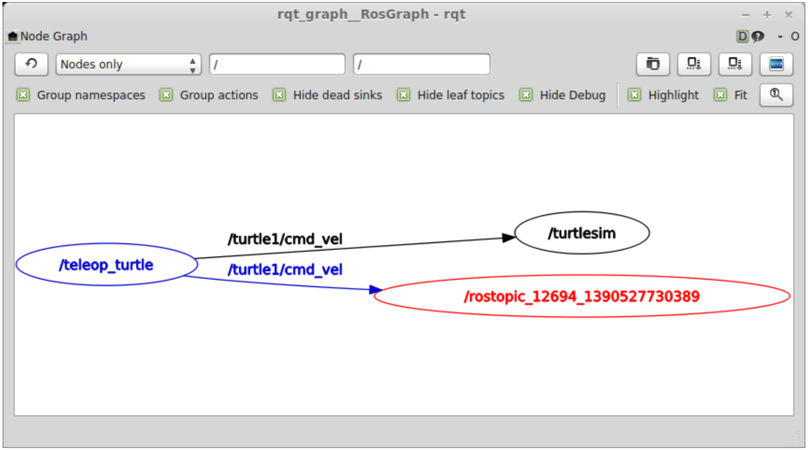
Try

```
rostopic list /turtle1/cmd_vel
```

Example (Would return)

```
---
linear:
  x: 2.0
  y: 0.0
  z: 0.0
angular:
  x: 0.0
  y: 0.0
  z: 0.0
---
```

rqt_graph


rqt_graph__RosGraph - rqt
Node Graph
Nodes only
/
/
Group namespaces
Group actions
Hide dead sinks
Hide leaf topics
Hide Debug
Highlight
Fit
/teleop_turtle
/turtle1/cmd_vel
/turtle1/cmd_vel
/turtlesim
/rostopic_12694_1390527730389

ROS Messages

- Communication on topics happens by sending ROS messages between nodes
- For the publisher `turtle_teleop_key` and subscriber `my_turtle` to communicate, the publisher and subscriber must send and receive the same type of message type
- A topic type is defined by the message type published on it

Usage

```
rostopic type [topic]
```

Try

```
rostopic type /turtle1/cmd_vel
```

Example (Would return)

```
geometry_msgs/Twist
```

rosmsg

- To get the message type of a topic, we use `rosmsg show`

Usage

```
rosmsg show [msg]
```

Try

```
rosmsg show turtlesim/Velocity
```

Example (Would return)

```
geometry_msgs/Vector3 linear
  float64 x
  float64 y
  float64 z
geometry_msgs/Vector3 angular
  float64 x
  float64 y
  float64 z
```

rostopic pub

- Now let's try and publish some commands to the turtle

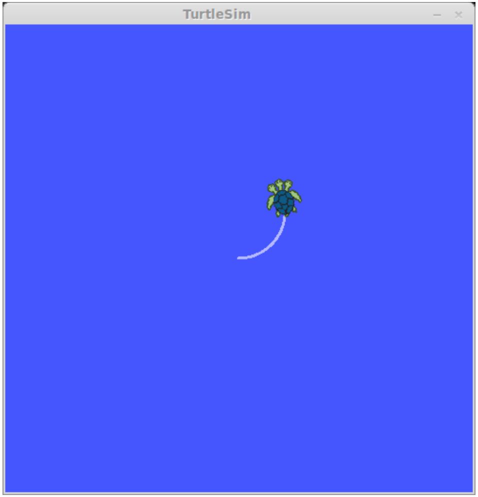
Usage

```
rostopic pub [topic] [msg_type] [args]
```

Try

```
rostopic pub -1 /turtle1/cmd_vel geometry_msgs/Twist --
 '[2.0, 0.0, 0.0]' '[0.0, 0.0, 1.8]'
```

turtlesim



Examine `rostopic pub`

- `rostopic pub`
  - This command will publish messages to a given topic
- `-1`
  - This option causes rostopic to only publish one message then exit
- `/turtle1/cmd_vel`
  - This is the name of the topic to publish to
- `geometry_msgs/Twist`
  - This is the message type to use when publishing the topic
- `--`
  - This tells the option parser that none of the following arguments are options
- `'[2.0, 0.0, 0.0]' '[0.0, 0.0, 1.8]'`

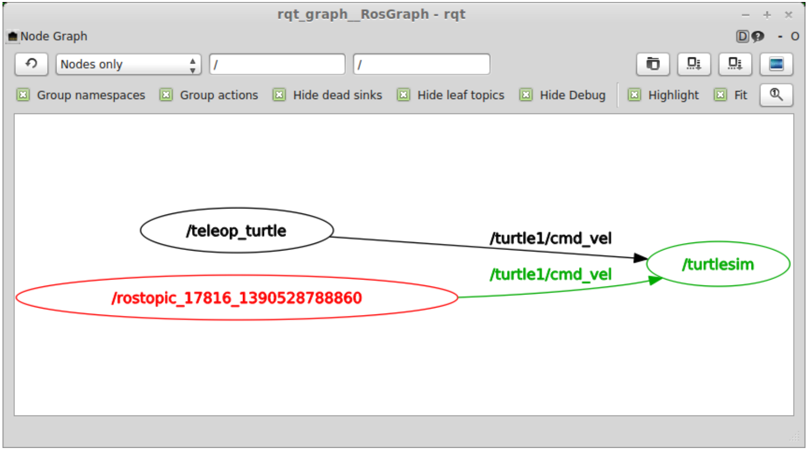
Continuous Publish

Try

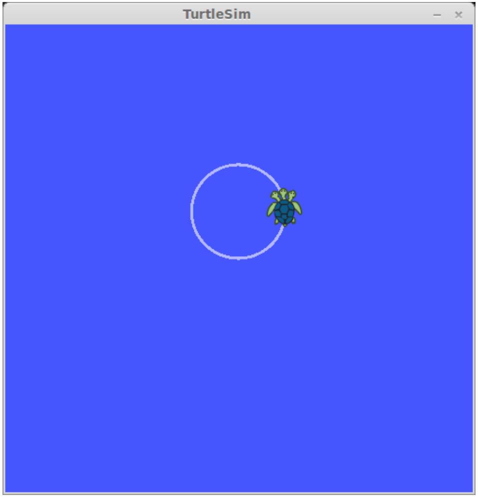
```
rostopic pub /turtle1/cmd_vel geometry_msgs/Twist -r 1 --
 '[2.0, 0.0, 0.0]' '[0.0, 0.0, 1.8]'
```

- This publishes the velocity commands at a rate of 1 Hz on the velocity topic

Continuous Publish in rqt_graph



Continuous Publish in `turtlesim`



rqt_plot

- rqt_plot displays a scrolling time plot of the data published on topics
- We'll use rqt_plot to plot the data being published on the /turtle1/pose topic

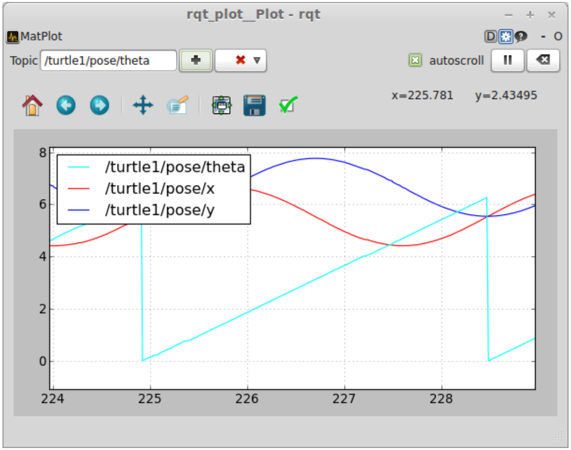
Try

```
rosrun rqt_plot rqt_plot
```

Add

```
/turtle1/pose/x
/turtle1/pose/y
/turtle1/pose/theta
```

rqt_plot


rqt_plot__Plot - rqt
MatPlot
Topic
/turtle1/pose/theta
autoscroll
x=225.781 y=2.43495
/turtle1/pose/theta
/turtle1/pose/x
/turtle1/pose/y
8
6
4
2
0
224
225
226
227
228