ROS Concepts & Commands

Seolyoung Jeong, Ph.D.

경북대학교 IT 대학

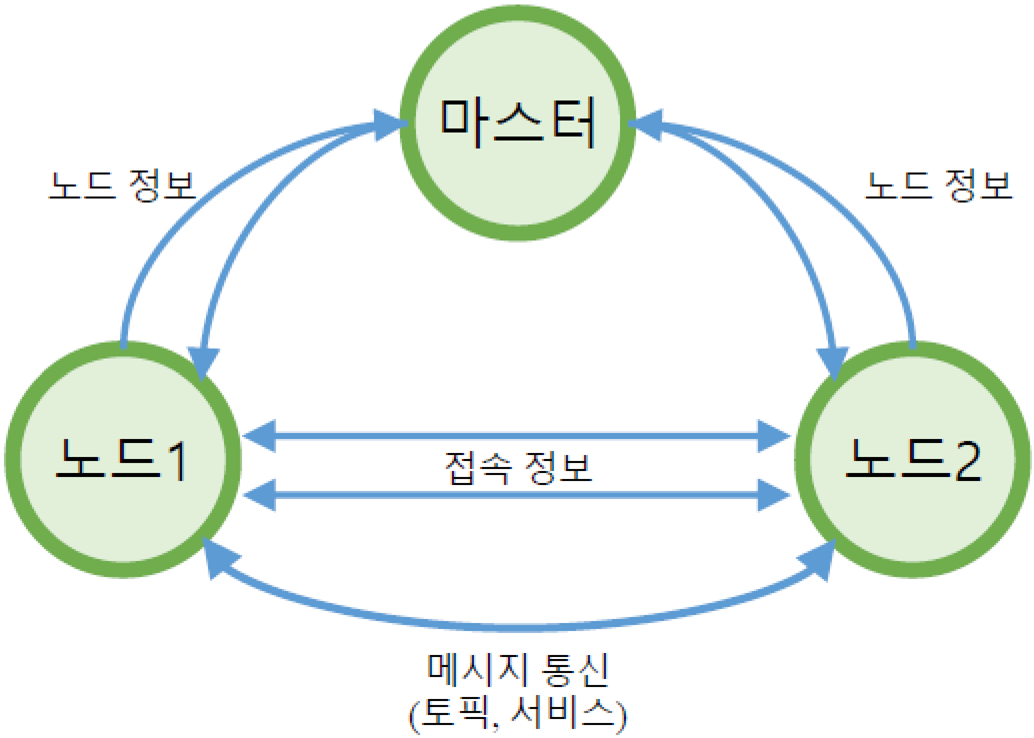
Contents

- **ROS Concepts**
  - ROS Message Communication
  - ROS Message
- **ROS Commands**
  - ROS Shell Commands
  - ROS Operation Commands
  - ROS Information Commands
- **ROS Simple Programming**
  - ROS File System
  - ROS Build System
  - Your first node program – “Hello, ROS!”

ROS Concepts

Message Communication

- **Communication between nodes**



ROS Master

- **Master**
  - provides naming and registration services to the rest of the nodes in the ROS system.
  - tracks publishers and subscribers to topics as well as services.
  - enable individual ROS nodes to locate one another. Once these nodes have located each other they communicate with each other peer-to-peer.
  - provides the Parameter Server.
  - provides an XMLRPC-based API, which ROS client libraries, such as roscpp and rospy, call to store and retrieve information

XMLRPC

- **XMLRPC (XML-Remote Procedure Call)**
  - a remote procedure call (RPC) protocol which uses XML to encode its calls and HTTP as a transport mechanism

- **RPC (Remote Procedure Call)**

  https://en.wikipedia.org/wiki/Remote_procedure_call

  - In distributed computing, a remote procedure call (RPC) is when a computer program causes a procedure (subroutine) to execute in a different address space (commonly on another computer on a shared network)
  - is coded as if it were a normal (local) procedure call, without the programmer explicitly coding the details for the remote interaction.

Environment Variables

Setup Environment

- export ROS_HOSTNAME=localhost
- export ROS_MASTER_URI=http://localhost:11311

ROS_MASTER_URI

- a required setting that tells nodes where they can locate the ROS Master
- set to the XML-RPC URI of the master
- URI (Uniform Resource Identifier, 통합자원식별자)
  - unambiguously identifies a particular resource
- Ex.
  - export ROS_MASTER_URI=http://mia:11311/

Environment Variables

- **ROS_HOSTNAME**
  - the declared network address of a ROS Node or tool
  - When a ROS component reports a URI to the master or other components, this value will be used.

  - only needed in situations where you have multiple addresses for a computer and need to force ROS to a particular one.
  - If the value is set to localhost, the ROS component will bind only to the loopback interface.

Message Communication

1. Run ROS Master

- command ‘$roscore’

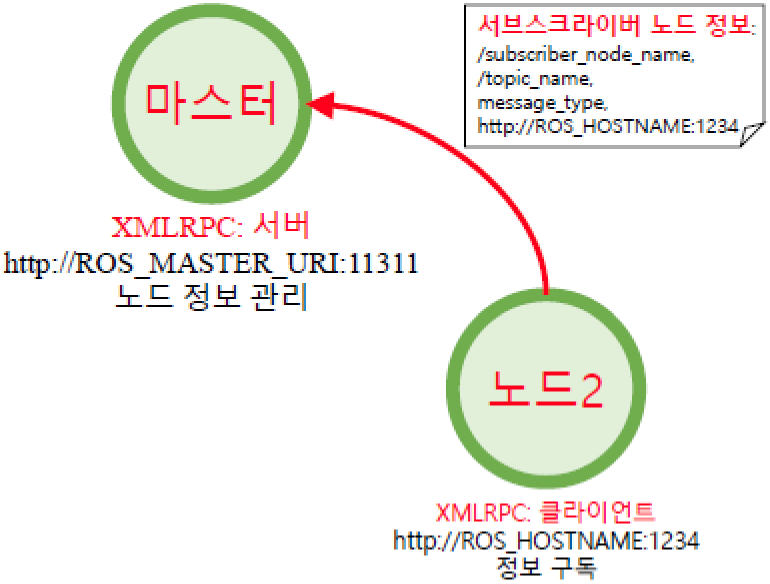
XMLRPC: 서버
http://ROS_MASTER_URI:11311
노드 정보 관리

- ROS Master (=Name Server)
  - Registration: node_name, topic, service, message_type, URI address/port
  - When requested, this information is reported to other nodes.

Message Communication

2. Run Node2 (Subscriber)

- command ‘$rosrun package_name node_name’

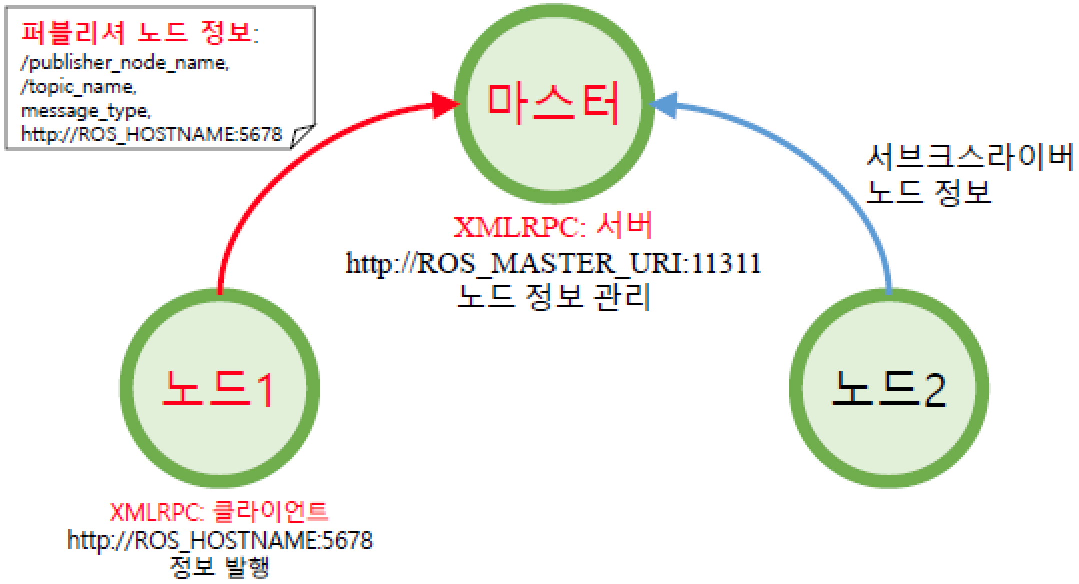


- node_name, topic_name, message_type, URI address/port
- using XMLRPC protocol

Message Communication

3. Run Node1 (Publisher)

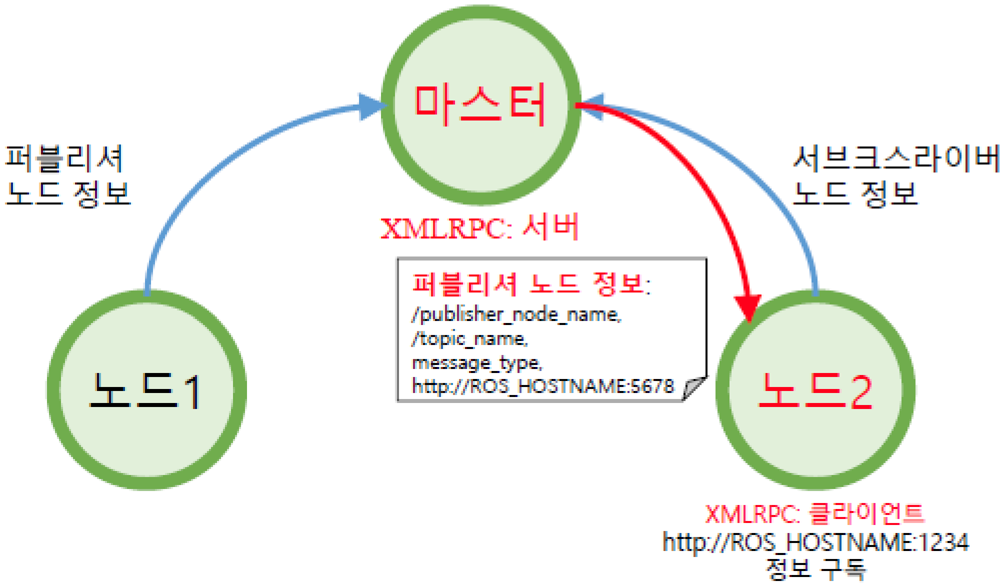
- command ‘$rosrun package_name node_name’



- node_name, topic_name, message_type, URI address/port
- using XMLRPC protocol

Message Communication

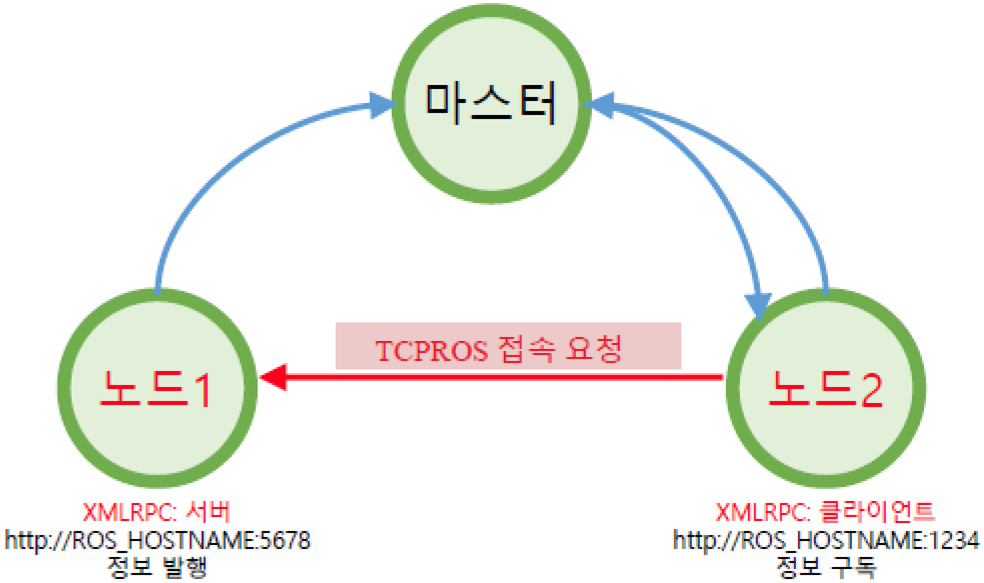
4. Master announce the information of publisher



- information of publisher that the subscriber wants to contact
- using XMLRPC protocol

Message Communication

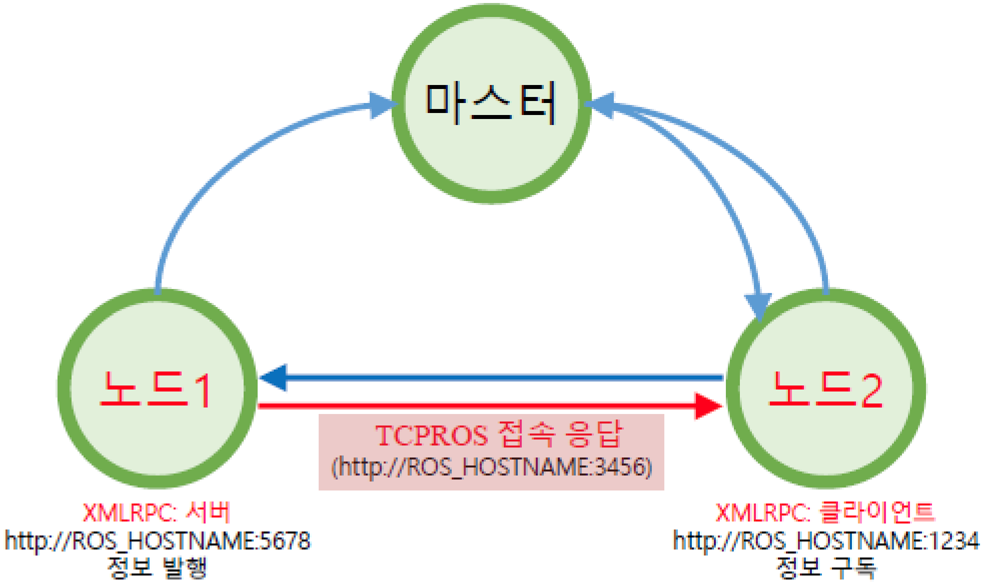
5. Subscriber request to contact



- directly request based on information received from the master
- using XMLRPC protocol

Message Communication

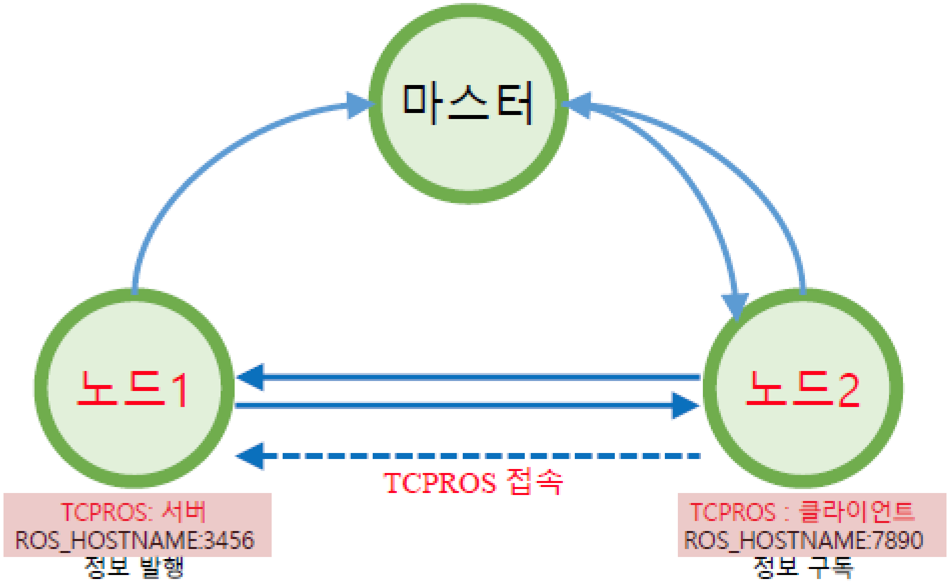
6. Publisher respond to subscriber



- Publisher’s TCPROS URI address/port
- using XMLRPC protocol

Message Communication

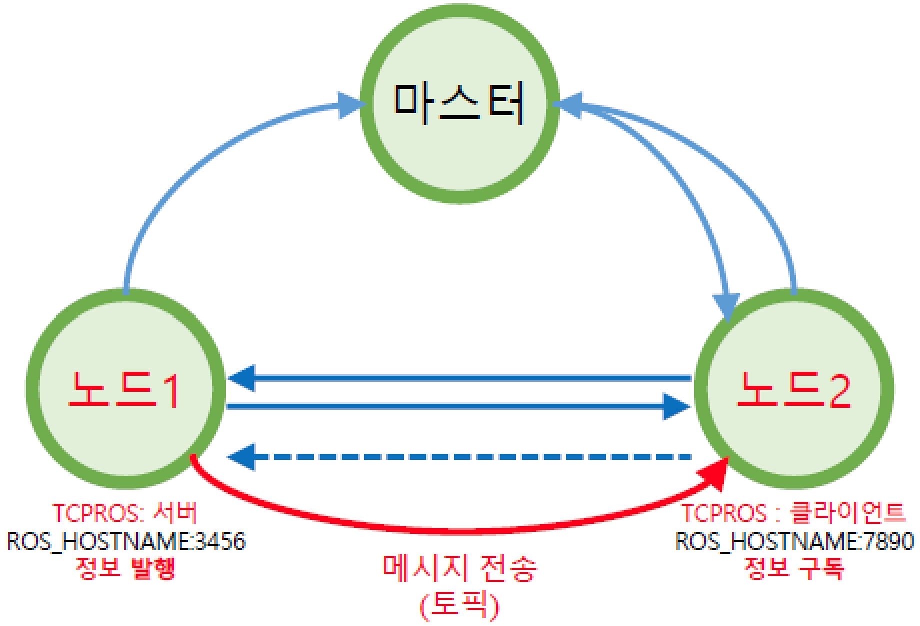
7. TCPROS connect



- directly connect between two nodes using TCPROS

Message Communication

8. Send message (topic)



Transport of Message on the ROS

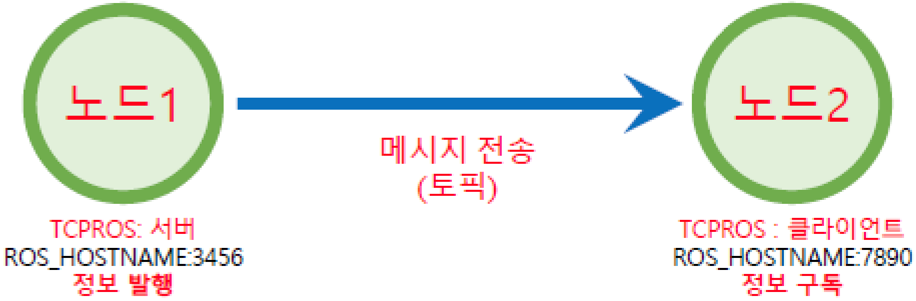
- **TCPROS**
  - a transport layer for ROS Messages and Services
  - It uses standard TCP/IP sockets for transporting message data.
  - Inbound connections are received via a TCP Server Socket with a header containing message data type and routing information.

- **UDPROS**
  - It uses standard UDP datagram packets to transport serialized message data.
  - The UDPROS transport is useful when latency is more important than reliable transport.
  - ex) streaming audio

Topic

Topics

- named buses over which nodes exchange messages
- anonymous publish/subscribe semantics
- multiple publishers and subscribers to a topic



Service

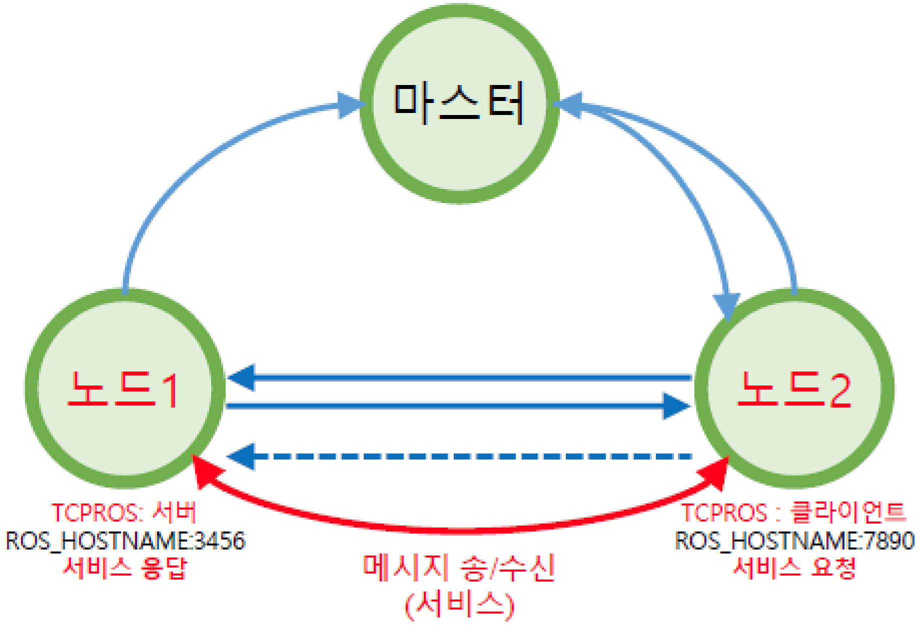
- **Topic**
  - The publish/subscribe model is a very flexible communication paradigm
  - its many-to-many one-way transport is not appropriate for RPC request/reply interactions

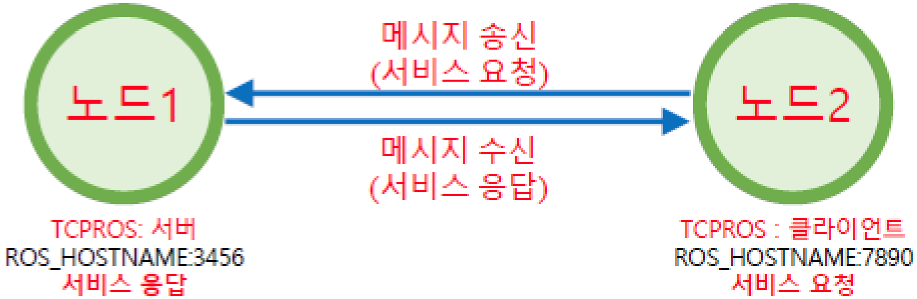
- **Service**
  - Request/reply is done via a Service, which is defined by a pair of messages: one for the request and one for the reply

Service

Service Request / Reply

- Service Server / Service Client





Understanding ROS Topics

Turtlesim package

- To understand ROS Topics, we use turtlesim.
- Run ROS master using roscore in terminal 1

```
$ roscore
```

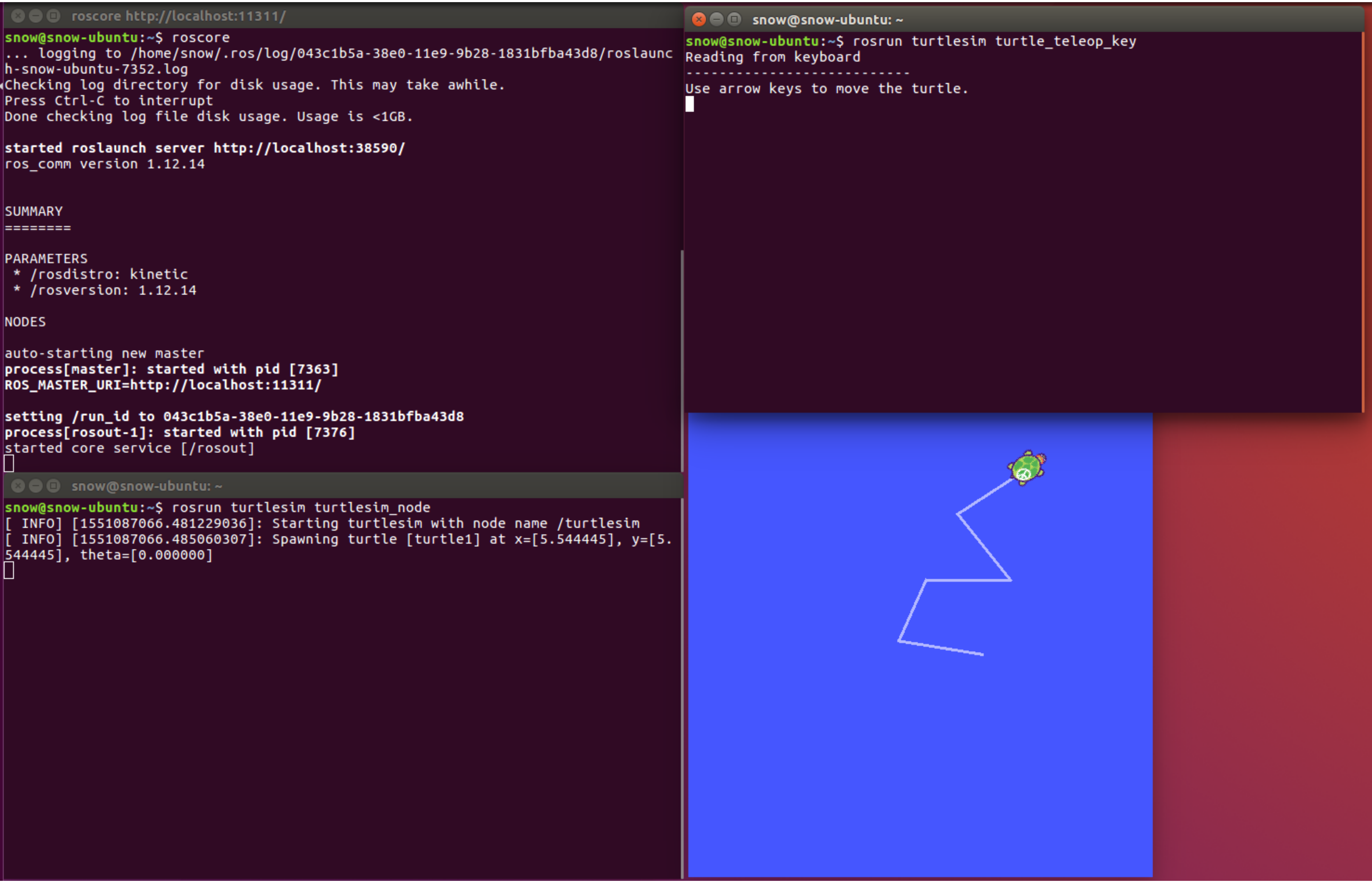
- Run turtlesim_node of turtlesim package in terminal 2

```
$ rosrun turtlesim turtlesim_node
```

- Run turtlesim_teleop_key of turtlesim package in terminal 3

```
$ rosrun turtlesim turtle_teleop_key
```

  - Now you can use the arrow keys of the keyboard to drive the turtle around.



```
snow@snow-ubuntu:~$ roscore
... logging to /home/snow/.ros/log/043c1b5a-38e0-11e9-9b28-1831bfba43d8/roslaunc
h-snow-ubuntu-7352.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://localhost:38590/
ros_comm version 1.12.14


SUMMARY
========

PARAMETERS
 * /rosdistro: kinetic
 * /rosversion: 1.12.14

NODES

auto-starting new master
process[master]: started with pid [7363]
ROS_MASTER_URI=http://localhost:11311/

setting /run_id to 043c1b5a-38e0-11e9-9b28-1831bfba43d8
process[rosout-1]: started with pid [7376]
started core service [/rosout]
```

```
snow@snow-ubuntu:~$ rosrun turtlesim turtlesim_node
[ INFO] [1551087066.481229036]: Starting turtlesim with node name /turtlesim
[ INFO] [1551087066.485060307]: Spawning turtle [turtle1] at x=[5.544445], y=[5.
544445], theta=[0.000000]
```

```
snow@snow-ubuntu:~$ rosrun turtlesim turtle_teleop_key
Reading from keyboard
---------------------------
Use arrow keys to move the turtle.
```

Understanding ROS Topics

- The turtlesim_node and the 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.
- Let's use rqt_graph which shows the nodes and topics currently running.
- Try

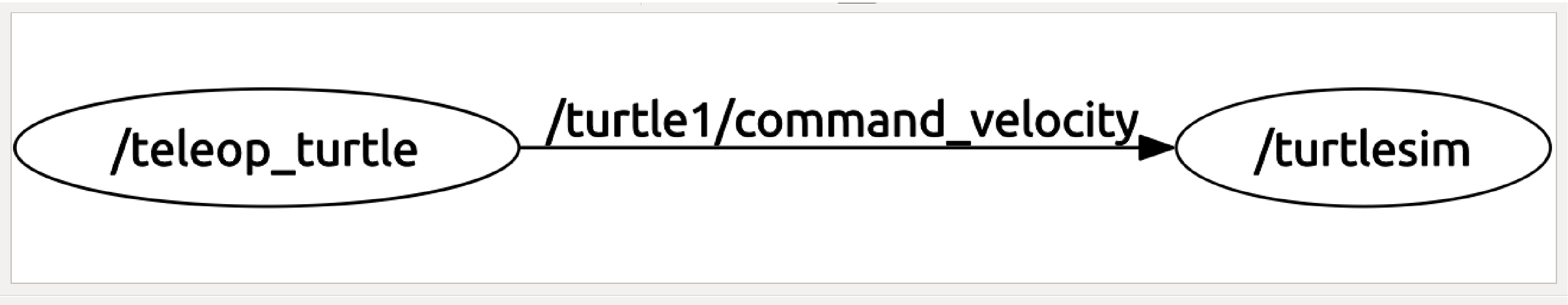
```
$ rqt_graph
```

or

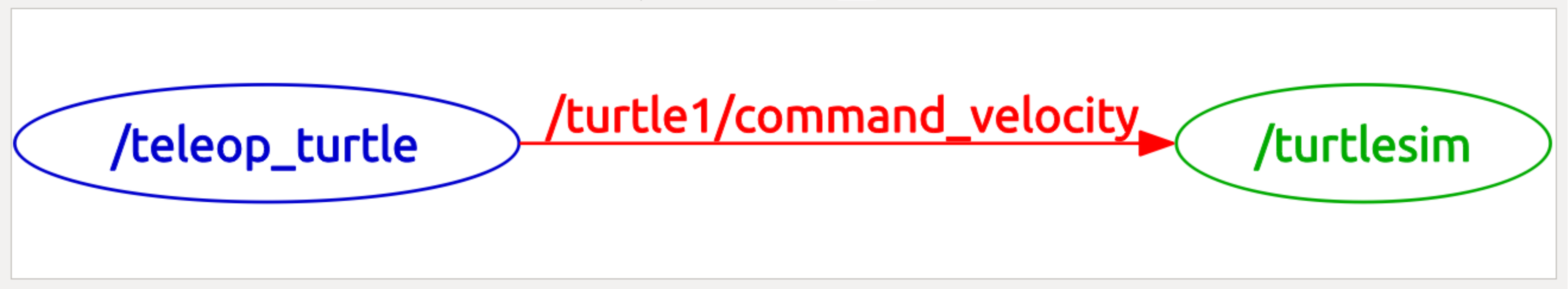
```
$ rosrun rqt_graph rqt_graph
```

Understanding ROS Topics

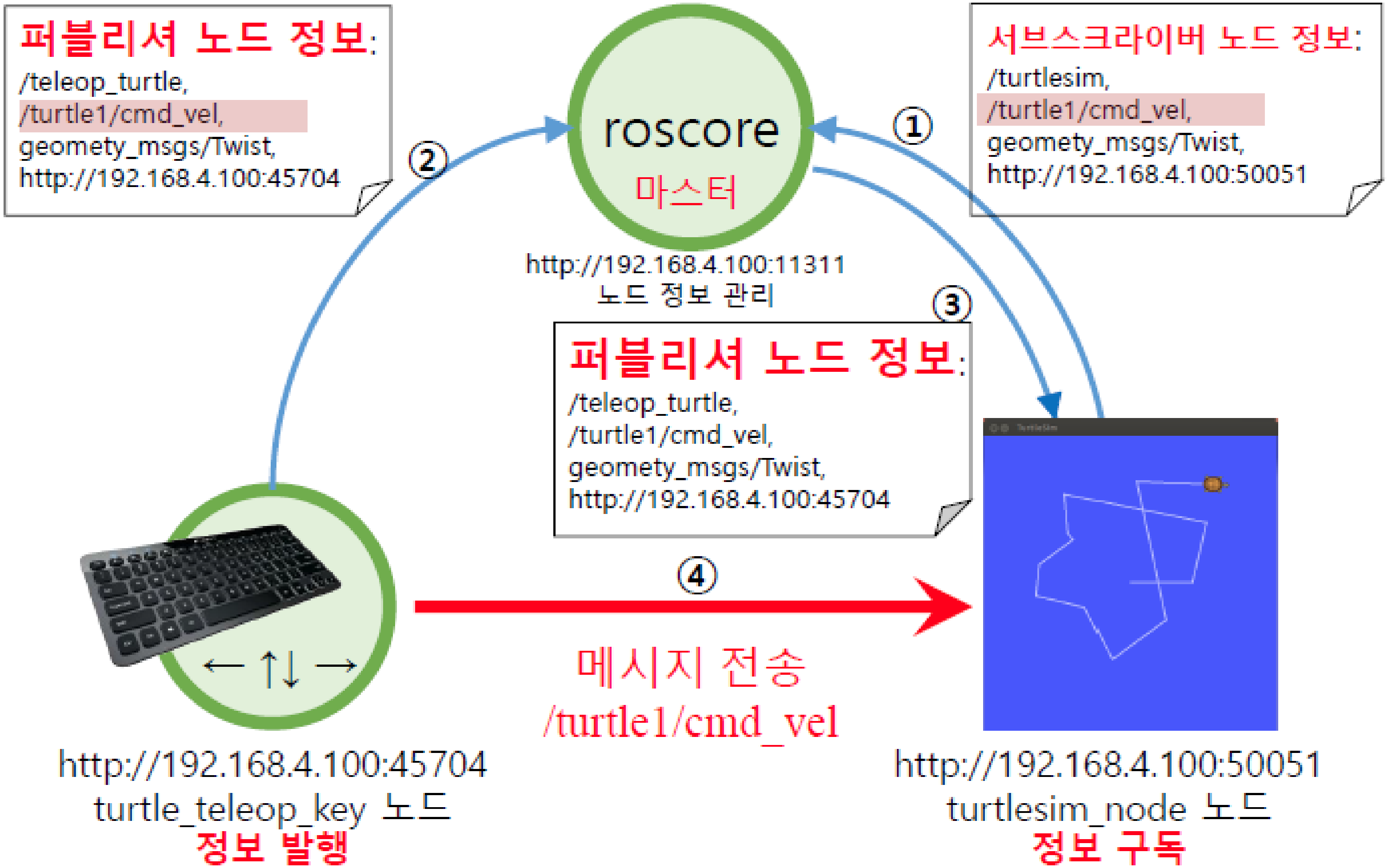
- **rqt_graph**



  - If you place your mouse over /turtle1/command_velocity it will highlight the ROS nodes (here blue and green) and topics (here red).



Turtlesim Topic



퍼블리셔 노드 정보:
/teleop_turtle,
/turtle1/cmd_vel,
geomety_msgs/Twist,
http://192.168.4.100:45704

②

roscore
마스터

http://192.168.4.100:11311
노드 정보 관리

①

서브스크라이버 노드 정보:
/turtlesim,
/turtle1/cmd_vel,
geomety_msgs/Twist,
http://192.168.4.100:50051

③

퍼블리셔 노드 정보:
/teleop_turtle,
/turtle1/cmd_vel,
geomety_msgs/Twist,
http://192.168.4.100:45704

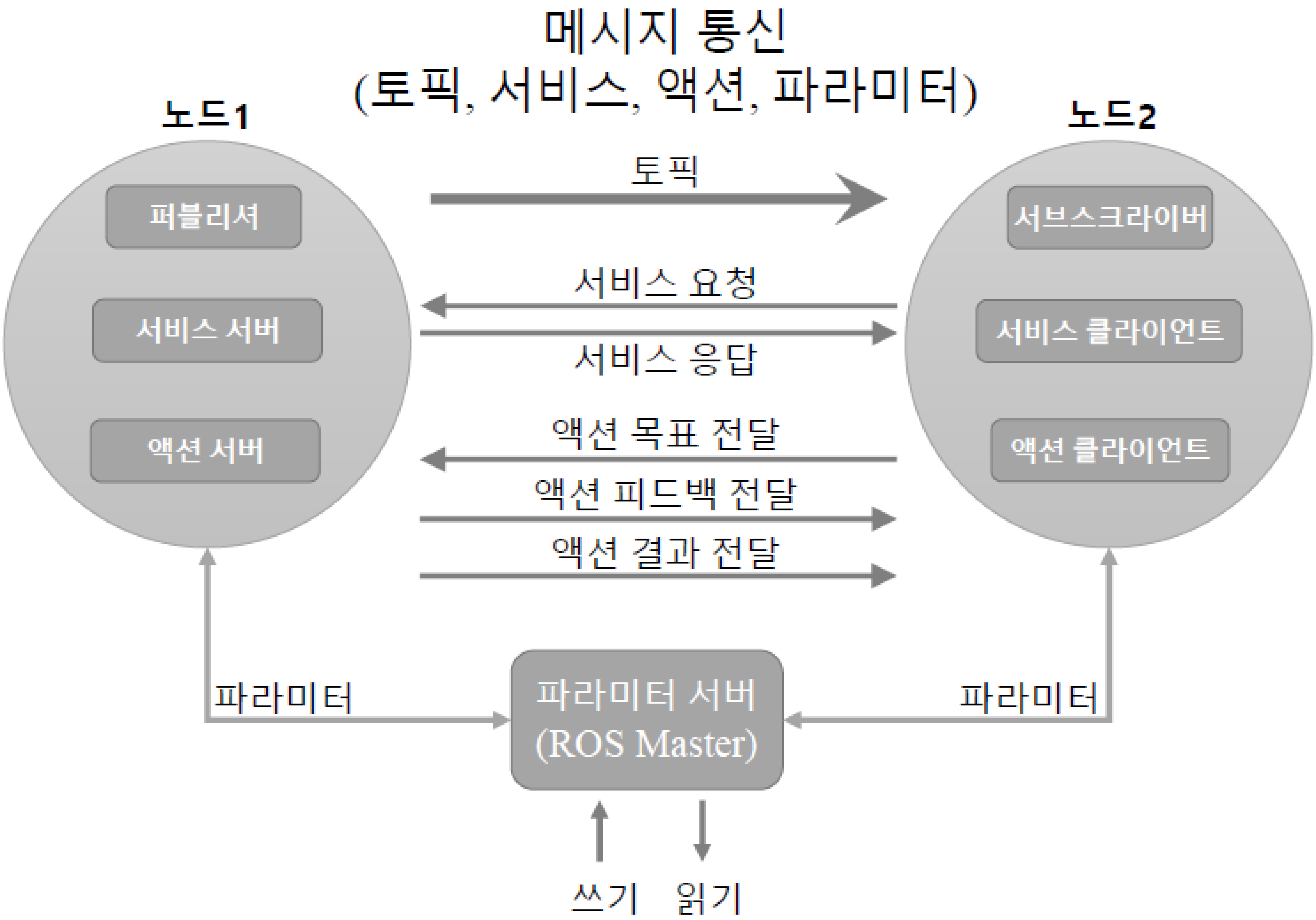
④

메시지 전송
/turtle1/cmd_vel

http://192.168.4.100:45704
turtle_teleop_key 노드
정보 발행

http://192.168.4.100:50051
turtlesim_node 노드
정보 구독

ROS Messages



ROS Messages

- **Message**
  - a simple data structure, comprising typed fields
  - Standard primitive types (integer, floating point, boolean, etc.) are supported, as are arrays of primitive types.
  - include arbitrarily nested structures and arrays

- **Nodes**
  - communicate with each other by publishing messages to topics.
  - exchange a request and response message as part of a ROS service call.

.msg file

- ROS uses a simplified messages description language for describing messages
  - This description makes it easy for ROS tools to automatically generate source code for the message type in several target languages.

- Message descriptions are stored in .msg files.
- .msg file
  - simple text files for specifying the data structure of a message
  - stored in the msg subdirectory of a package (msg/ subdirectory)

Message Description Specification

http://wiki.ros.org/msg

Message Description

- a list of data field descriptions and constant definitions on separate lines.

Fields : type + name

```
fieldtype1 fieldname1
fieldtype2 fieldname2
fieldtype3 fieldname3
```

- For example :

```
int32 x
int32 y
```

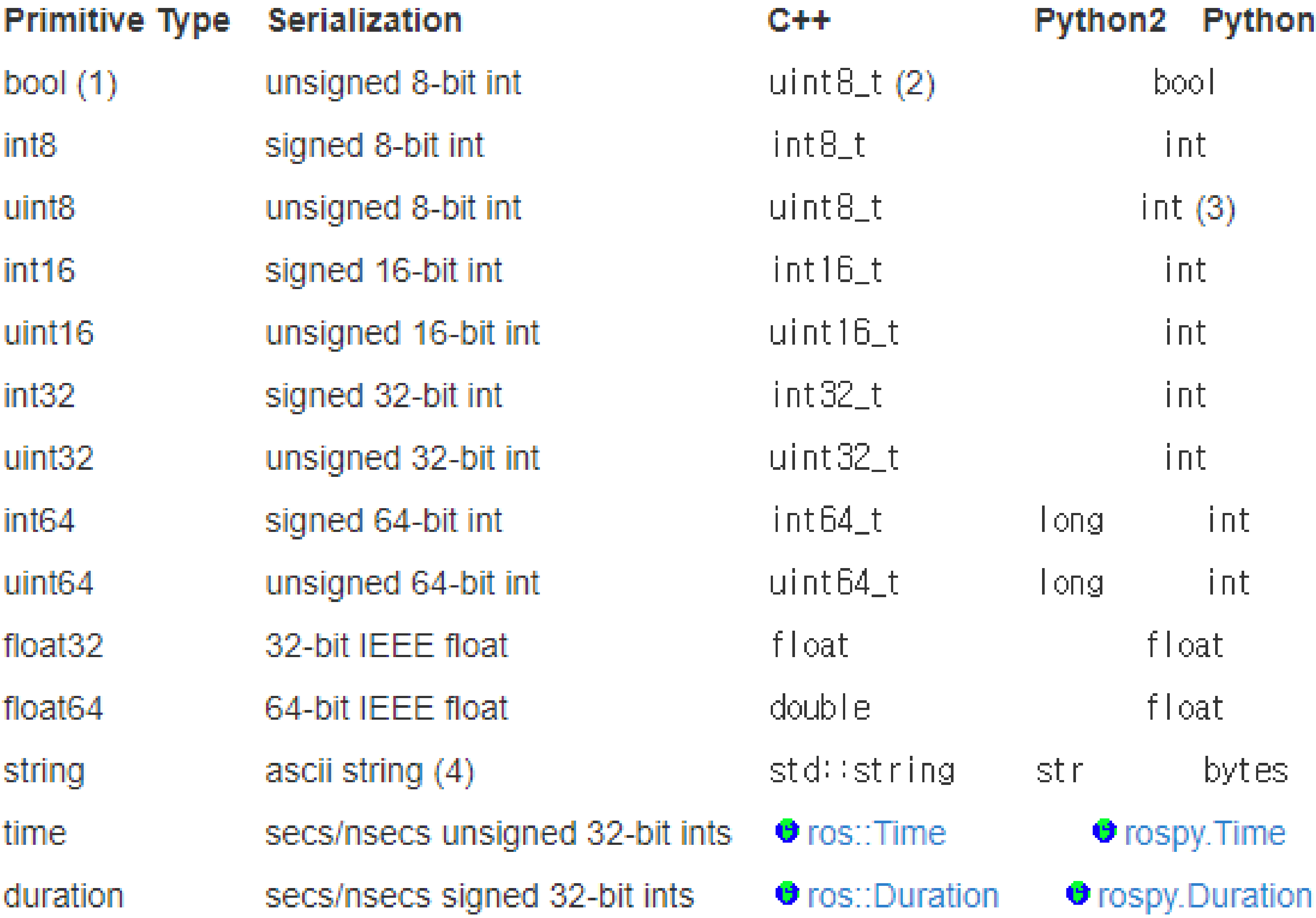
ROS Messages

- **Field Types**
  - built-in type, such as "float32 pan" or "string name"
  - fixed- or variable-length arrays (lists) of the above, such as "float32[] ranges" or "Point32[10] points“
  - names of Message descriptions defined on their own, such as "geometry_msgs/PoseStamped"
  - special Header type, which maps to std_msgs/Header

  - must not use the names of built-in types or Header when constructing own message types.

ROS Messages

- **Built-in types**



Primitive Type	Serialization	C++	Python2	Python3
bool (1)	unsigned 8-bit int	uint8_t (2)	bool	
int8	signed 8-bit int	int8_t	int	
uint8	unsigned 8-bit int	uint8_t	int (3)	
int16	signed 16-bit int	int16_t	int	
uint16	unsigned 16-bit int	uint16_t	int	
int32	signed 32-bit int	int32_t	int	
uint32	unsigned 32-bit int	uint32_t	int	
int64	signed 64-bit int	int64_t	long	int
uint64	unsigned 64-bit int	uint64_t	long	int
float32	32-bit IEEE float	float	float	
float64	64-bit IEEE float	double	float	
string	ascii string (4)	std::string	str	bytes
time	secs/nsecs unsigned 32-bit ints	ros::Time	rospy.Time	
duration	secs/nsecs signed 32-bit ints	ros::Duration	rospy.Duration	

ROS Messages

Header

- ROS provides the special Header type to provide a general mechanism for setting frame IDs for libraries.
- Header is not a built-in type.
  (it's defined in std_msgs/msg/Header.msg)

- .msg file example:

```
Header header
```

  It will be resolved as ‘std_msgs/Header’.

- Header.msg:

```
#Standard metadata for higher-level flow data types
#sequence ID: consecutively increasing ID
uint32 seq
#Two-integer timestamp that is expressed as:
# * stamp.secs: seconds (stamp_secs) since epoch
# * stamp.nsecs: nanoseconds since stamp_secs
# time-handling sugar is provided by the client library
time stamp
#Frame this data is associated with
string frame_id
```

ROS Messages

Constants

- Each constant definition is like a field description, except that it also assigns a value.
- This value assignment is indicated by use of an equal '=' sign, e.g.

```
constanttype1 CONSTANTNAME1=constantvalue1
constanttype2 CONSTANTNAME2=constantvalue2
```

- For example:

```
int32 X=123
int32 Y=-123
string FOO=foo
string EXAMPLE="#comments" are ignored, and leading and trailing whitespace removed
```

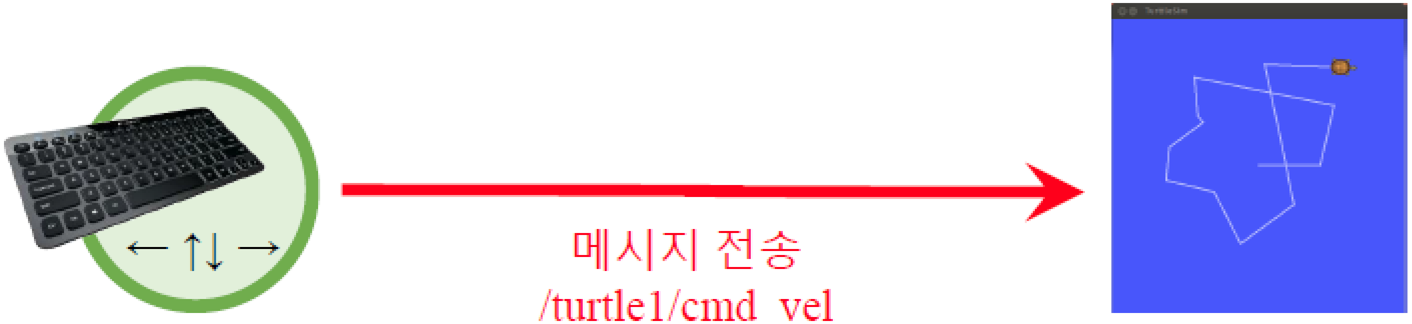
ROS Messages

- **common_msgs**
  - widely used by other ROS packages

  - actions (actionlib_msgs)
  - diagnostics (diagnostic_msgs)
  - geometric primitives (geometry_msgs)
  - robot navigation (nav_msgs)
  - common sensors (sensor_msgs)

ROS Messages

- **Example : geometry_msgs/Twist**



http://192.168.4.100:45704
turtle_teleop_key 노드
정보 발행

메시지 전송
/turtle1/cmd_vel
(geometry_msgs/Twist)

http://192.168.4.100:50051
turtlesim_node 노드
정보 구독

[geometry_msgs/Twist]

Vector3 linear
Vector3 angular

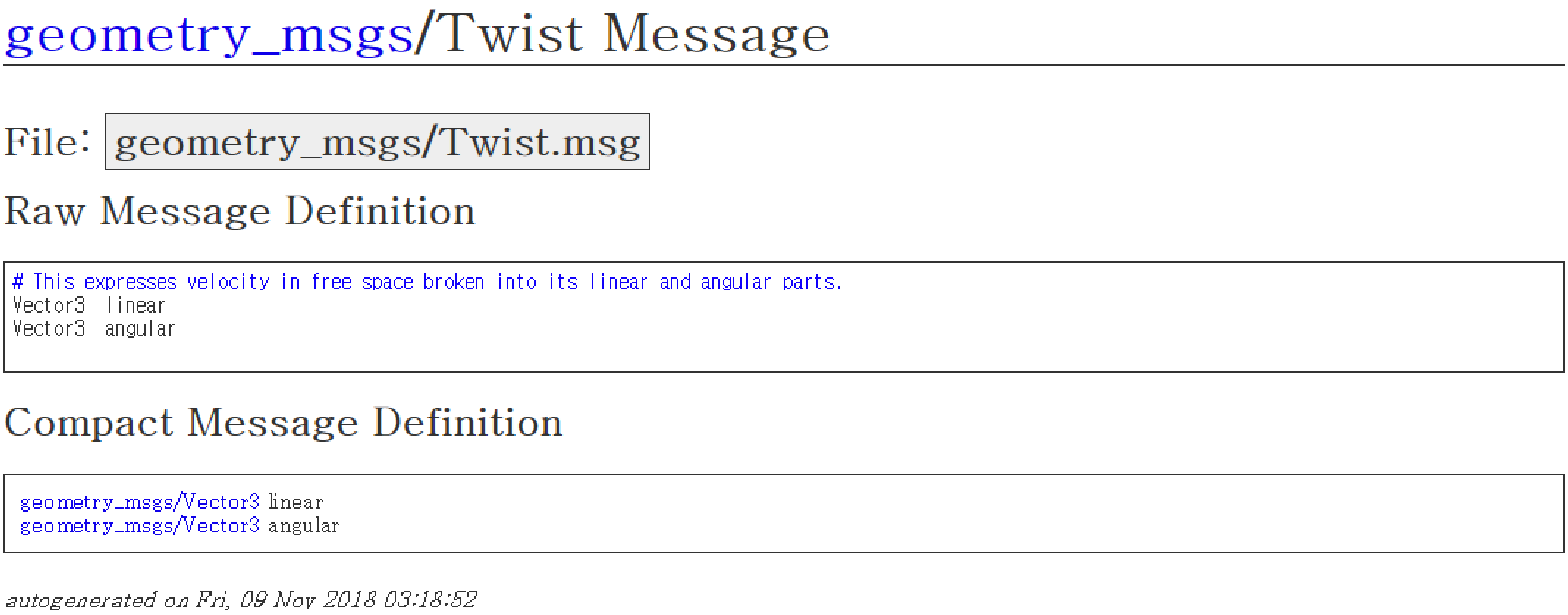
[geometry_msgs/Vector3]
float64 x
float64 y
float64 z

[geometry_msgs/Vector3]
float64 x
float64 y
float64 z

ROS Messages

- **geometry_msgs/Twist**

http://docs.ros.org/api/geometry_msgs/html/msg/Twist.html



geometry_msgs/Twist Message

File: geometry_msgs/Twist.msg

Raw Message Definition

```
# This expresses velocity in free space broken into its linear and angular parts.
Vector3  linear
Vector3  angular
```

Compact Message Definition

```
geometry_msgs/Vector3 linear
geometry_msgs/Vector3 angular
```

autogenerated on Fri, 09 Nov 2018 03:18:52

.srv file

- ROS uses a simplified service description language ("srv") for describing ROS service types.
- Service descriptions are stored in .srv files in the srv/ subdirectory of a package.
- consists of a request and a response msg type, separated by '---'

request messages

response messages

```
string str
---
string str
```

- For example:
  - sensor_msgs/SetCameraInfo.srv

```
sensor_msgs/CameraInfo camera_info
---
bool success
string status_message
```

http://docs.ros.org/melodic/api/sensor_msgs/html/srv/SetCameraInfo.html

ROS Commands

ROS Commands

https://wiki.ros.org/ROS/CommandLineTools

ROS Shell Commands

- roscd
- rosls

ROS Operation Commands

- roscore
- rosrun
- roslaunch

ROS Information Commands

- rospack
- rosnode
- rostopic
- rosmsg
- rosservice
- rosparam

ROS Shell Commands

roscd

- allows us to ‘cd’ directly to a package or stack
- Usage:

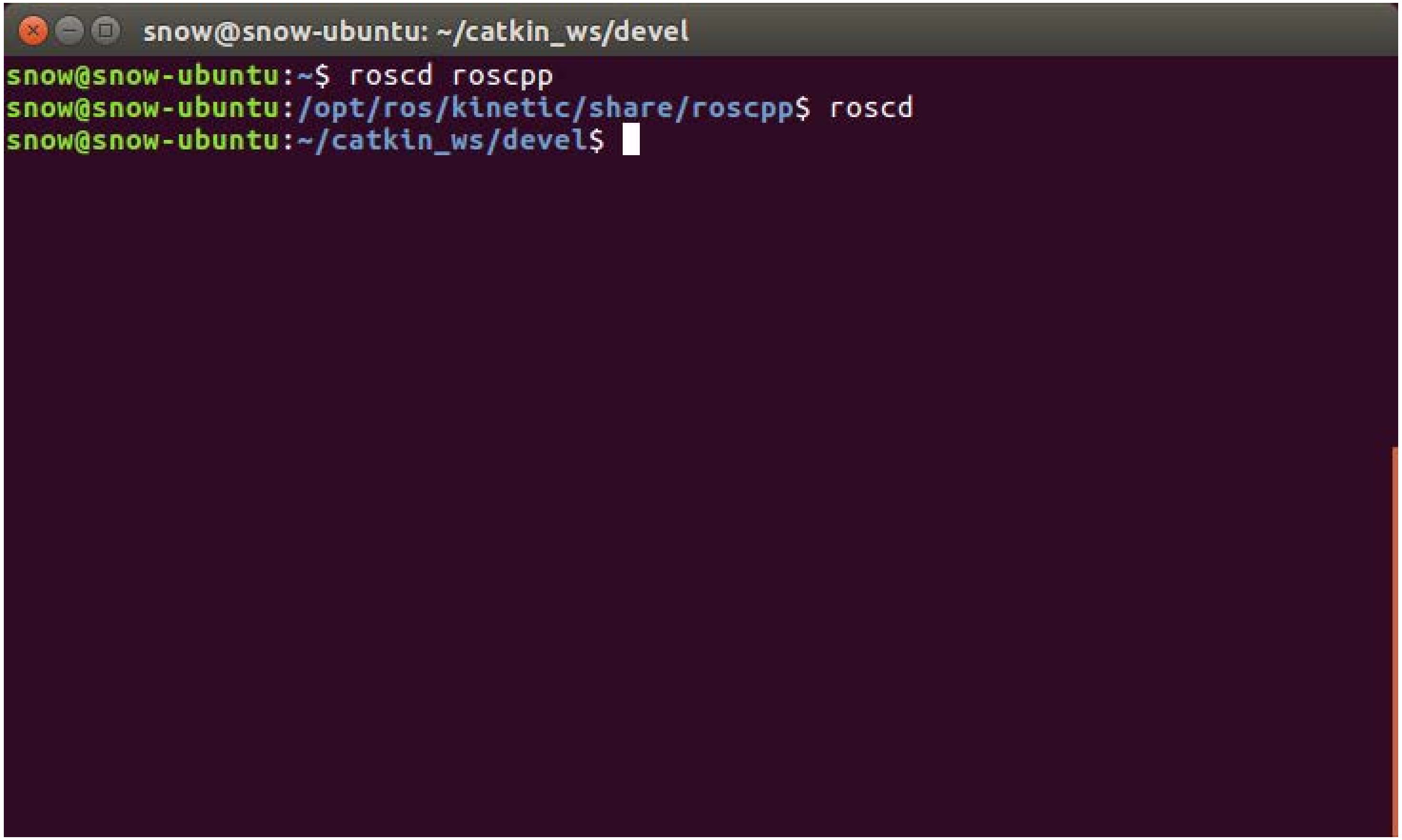
```
$ roscd locationname[/subdir]
$ roscd packagename
```

- Example:

```
$ roscd roscpp/include
$ roscd turtlesim
```

- Try
  - change current directory into ‘roscpp’
  - change current directory into ‘/catkin_ws/devel’

```
$ roscd roscpp
$ roscd
```



snow@snow-ubuntu: ~/catkin_ws/devel

```
snow@snow-ubuntu:~$ roscd roscpp
snow@snow-ubuntu:/opt/ros/kinetic/share/roscpp$ roscd
snow@snow-ubuntu:~/catkin_ws/devel$ 
```

ROS Shell Commands

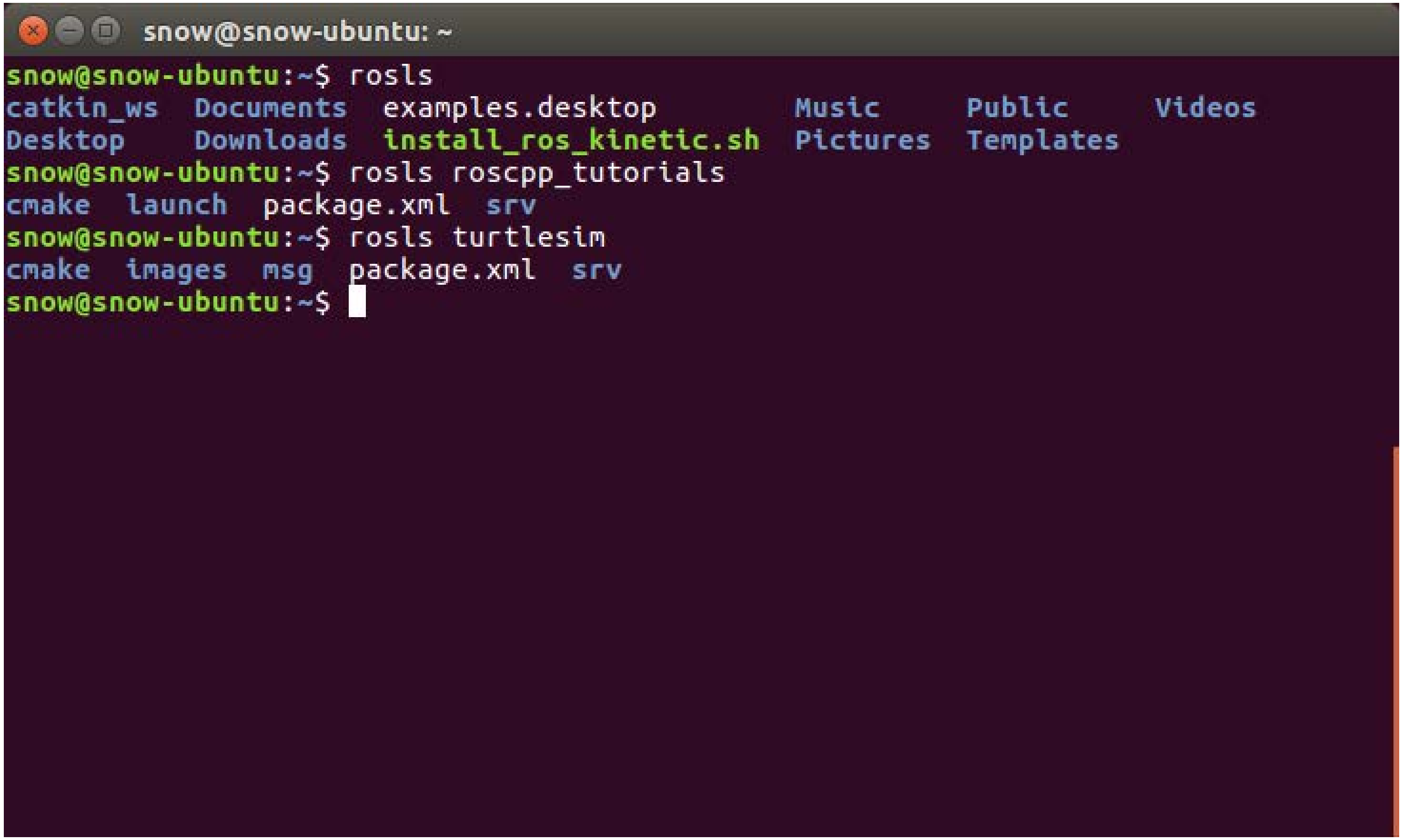
rosls

- allows us to view the contents of a package, stack, or location.
- Example:

```
$ rosls roscpp
$ rosls roscpp/include/ros
$ rosls turtlesim
```

- Try
  - Print out a list of files at ‘/catkin_ws/devel’
  - Print out a list of files with a package name

```
$ rosls /catkin_ws/devel
$ rosls roscpp_tutorials
$ rosls turtlesim
```

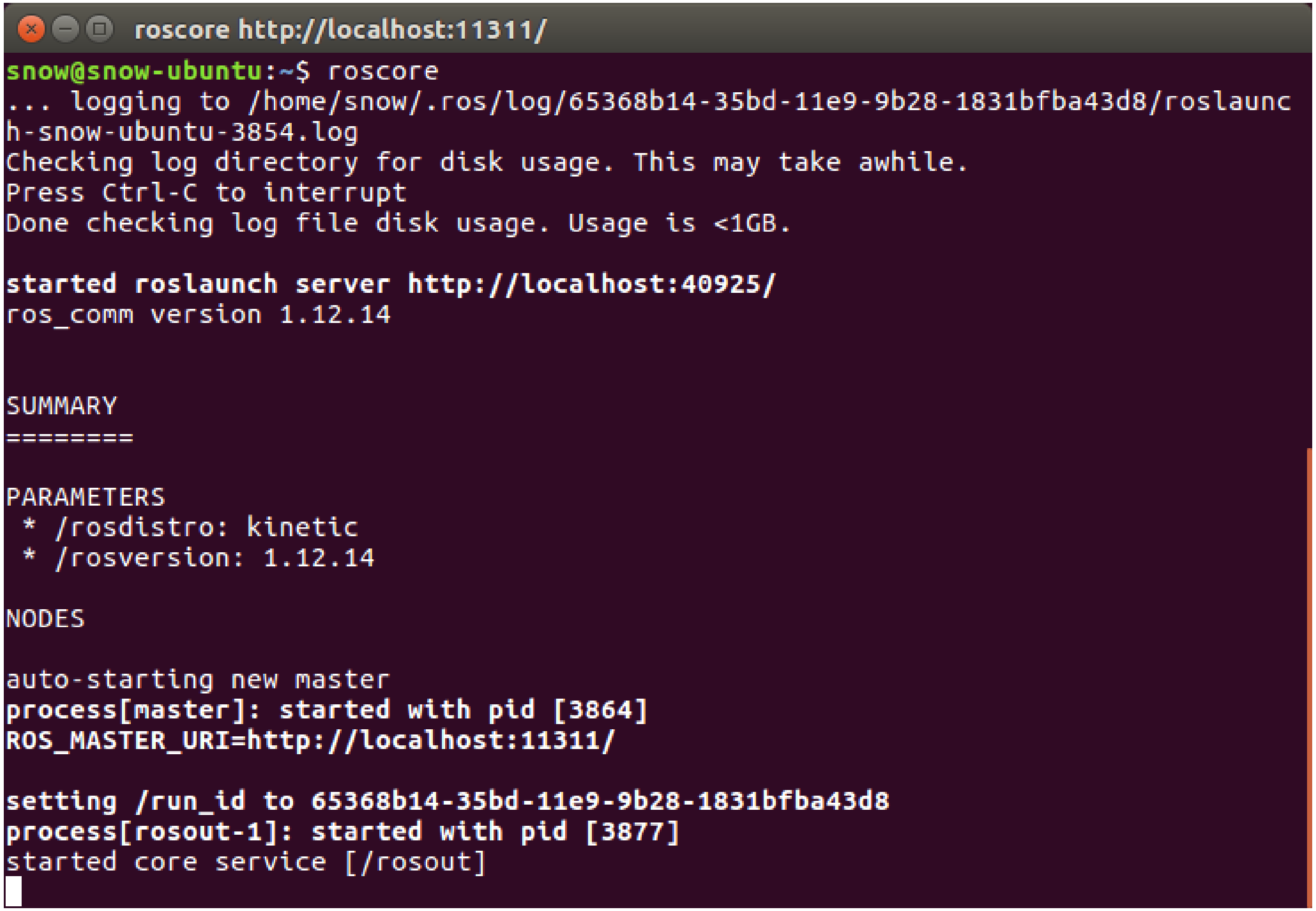


```
snow@snow-ubuntu:~$ rosls
catkin_ws  Documents  examples.desktop          Music     Public     Videos
Desktop    Downloads  install_ros_kinetic.sh   Pictures  Templates
snow@snow-ubuntu:~$ rosls roscpp_tutorials
cmake  launch  package.xml  srv
snow@snow-ubuntu:~$ rosls turtlesim
cmake  images  msg  package.xml  srv
snow@snow-ubuntu:~$ 
```

ROS Operation Commands

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

```
$ roscore
```



roscore http://localhost:11311/

```
snow@snow-ubuntu:~$ roscore
... logging to /home/snow/.ros/log/65368b14-35bd-11e9-9b28-1831bfba43d8/roslaunc
h-snow-ubuntu-3854.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://localhost:40925/
ros_comm version 1.12.14


SUMMARY
========

PARAMETERS
 * /rosdistro: kinetic
 * /rosversion: 1.12.14

NODES

auto-starting new master
process[master]: started with pid [3864]
ROS_MASTER_URI=http://localhost:11311/

setting /run_id to 65368b14-35bd-11e9-9b28-1831bfba43d8
process[rosout-1]: started with pid [3877]
started core service [/rosout]
```

ROS Operation Commands

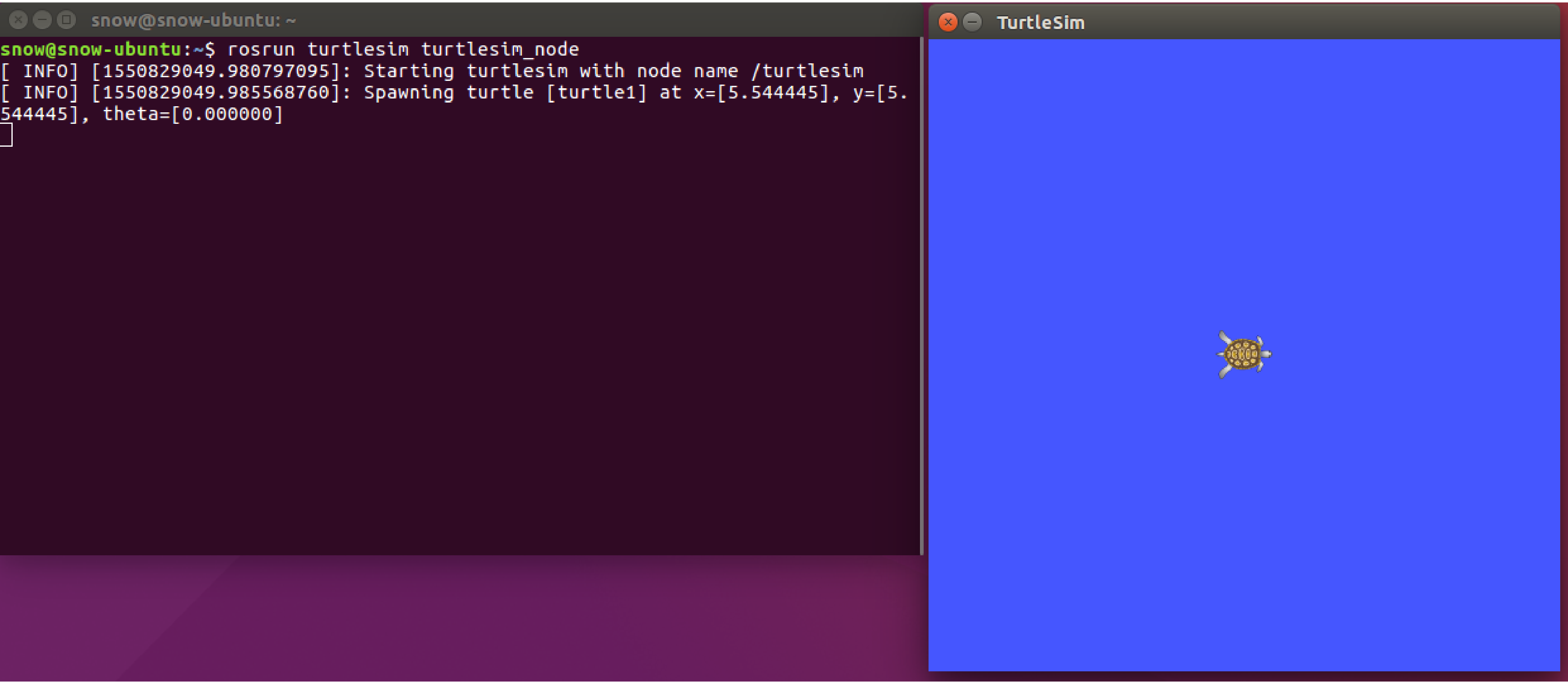
rosrun

- allows us to run an executable in an arbitrary package (without having to know the package path)
- Usage:

```
$ rosrun package_name node_name
```

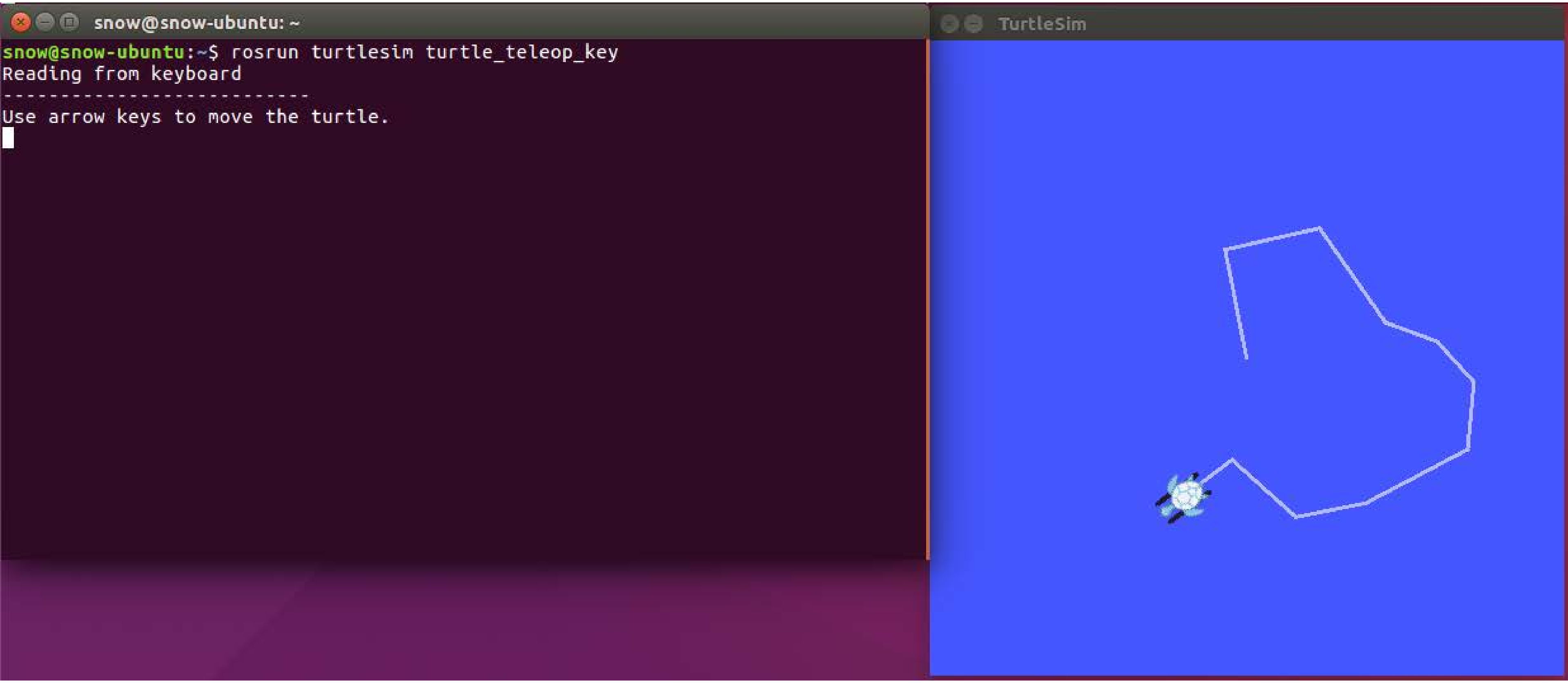
- Try:
  - Running the turtlesim_node in the tuetlesim package
  - ROS Installation & Simple Test (new Terminal)

```
$ rosrun turtlesim turtlesim_node
```



```
snow@snow-ubuntu:~$ rosrun turtlesim turtlesim_node
[ INFO] [1550829049.980797095]: Starting turtlesim with node name /turtlesim
[ INFO] [1550829049.985568760]: Spawning turtle [turtle1] at x=[5.544445], y=[5.
544445], theta=[0.000000]
```

```
$ rosrun turtlesim turtle_teleop_key
```



```
snow@snow-ubuntu:~$ rosrun turtlesim turtle_teleop_key
Reading from keyboard
---------------------------
Use arrow keys to move the turtle.
```

ROS Operation Commands

- **roslaunch**
  - a tool for easily launching multiple ROS nodes (locally and remotely)
  - Usage: `$ roslaunch package_name file.launch`
  - a set of nodes from an XML configuration file (.launch)

```
<launch>
  <!-- local machine already has a definition by default.
       This tag overrides the default definition with
       specific ROS_ROOT and ROS_PACKAGE_PATH values -->
  <machine name="local_alt" address="localhost" default="true" ros-root="/u/user/ros/ros/" ros-package-path="/u/user/ros/ros-pkg" />
  <!-- a basic listener node -->
  <node name="listener-1" pkg="rospy_tutorials" type="listener" />
  <!-- pass args to the listener node -->
  <node name="listener-2" pkg="rospy_tutorials" type="listener" args="-foo arg2" />
  <!-- a respawn-able listener node -->
  <node name="listener-3" pkg="rospy_tutorials" type="listener" respawn="true" />
  <!-- start listener node in the 'wg1' namespace -->
  <node ns="wg1" name="listener-wg1" pkg="rospy_tutorials" type="listener" respawn="true" />
  <!-- start a group of nodes in the 'wg2' namespace -->
  <group ns="wg2">
    <!-- remap applies to all future statements in this scope. -->
    <remap from="chatter" to="hello"/>
    <node pkg="rospy_tutorials" type="listener" name="listener" args="--test" respawn="true" />
    <node pkg="rospy_tutorials" type="talker" name="talker">
      <!-- set a private parameter for the node -->
      <param name="talker_1_param" value="a value" />
      <!-- nodes can have their own remap args -->
      <remap from="chatter" to="hello-1"/>
      <!-- you can set environment variables for a node -->
      <env name="ENV_EXAMPLE" value="some value" />
    </node>
  </group>
</launch>
```

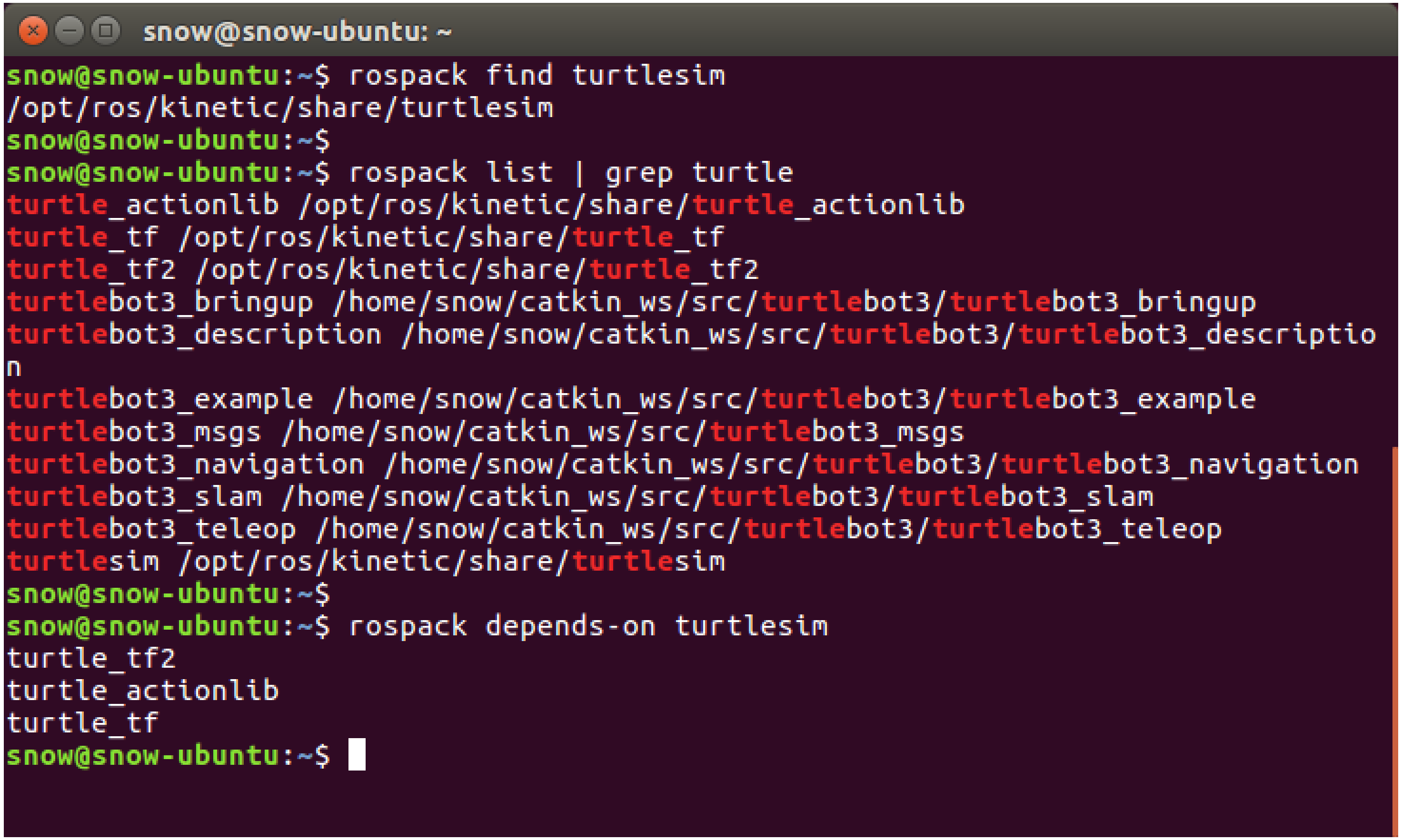
ROS Information Commands

- **rospack**
  - retrieving information about ROS packages available on the filesystem
  - Usage:

    ```
    $ rospack <command> <options> <package_name>
    ```

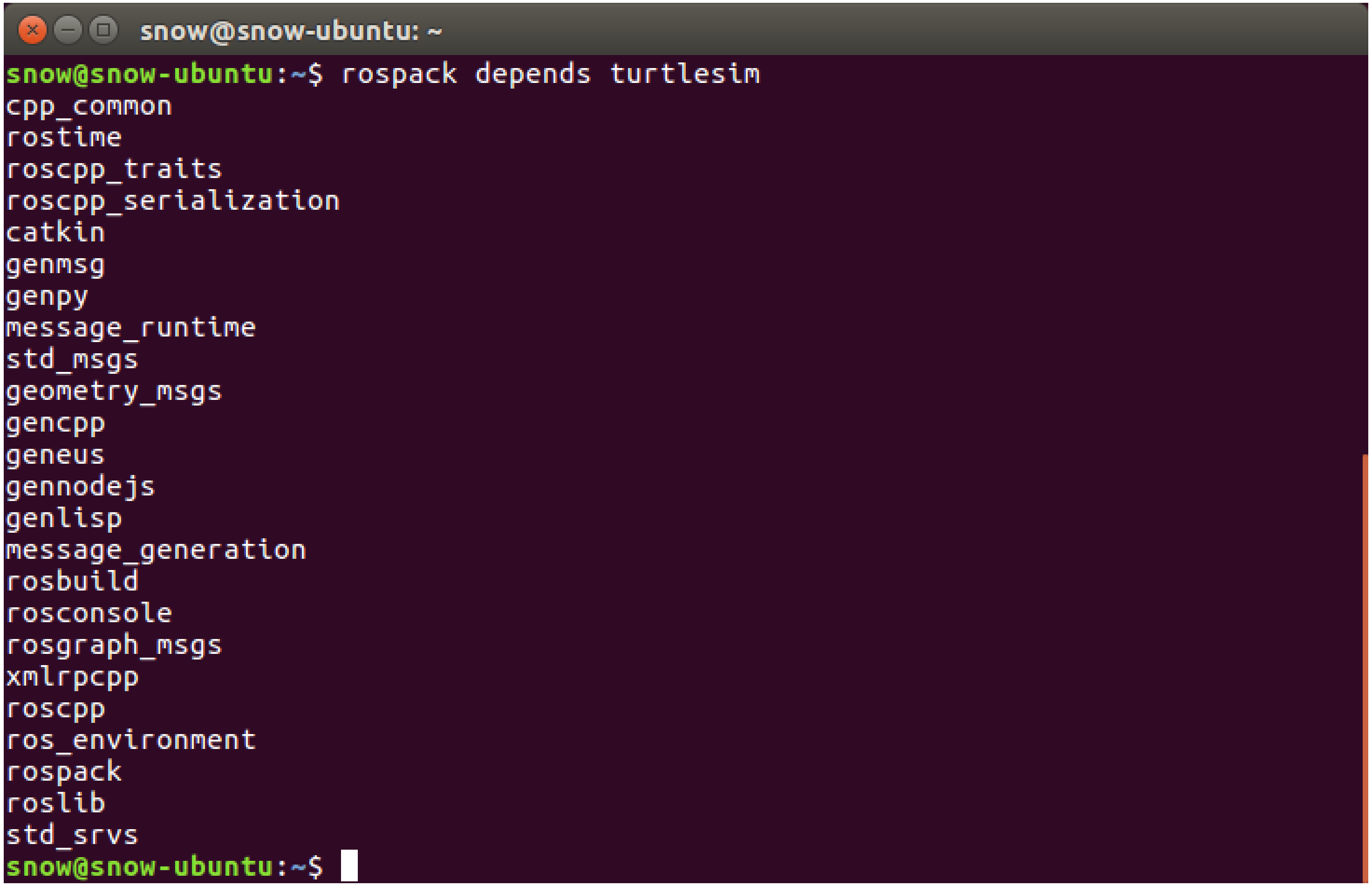
  - Try:
    - Find the ‘turtlesim’ package in Ubuntu filesystem
    - Print out a list of current ROS packages
    - ...

```
$ rospack find turtlesim
$ rospack list
$ rospack list | grep turtle
$ rospack depends-on turtlesim
$ rospack depends turtlesim
```



```
snow@snow-ubuntu:~$ rospack find turtlesim
/opt/ros/kinetic/share/turtlesim
snow@snow-ubuntu:~$
snow@snow-ubuntu:~$ rospack list | grep turtle
turtle_actionlib /opt/ros/kinetic/share/turtle_actionlib
turtle_tf /opt/ros/kinetic/share/turtle_tf
turtle_tf2 /opt/ros/kinetic/share/turtle_tf2
turtlebot3_bringup /home/snow/catkin_ws/src/turtlebot3/turtlebot3_bringup
turtlebot3_description /home/snow/catkin_ws/src/turtlebot3/turtlebot3_descriptio
n
turtlebot3_example /home/snow/catkin_ws/src/turtlebot3/turtlebot3_example
turtlebot3_msgs /home/snow/catkin_ws/src/turtlebot3_msgs
turtlebot3_navigation /home/snow/catkin_ws/src/turtlebot3/turtlebot3_navigation
turtlebot3_slam /home/snow/catkin_ws/src/turtlebot3/turtlebot3_slam
turtlebot3_teleop /home/snow/catkin_ws/src/turtlebot3/turtlebot3_teleop
turtlesim /opt/ros/kinetic/share/turtlesim
snow@snow-ubuntu:~$
snow@snow-ubuntu:~$ rospack depends-on turtlesim
turtle_tf2
turtle_actionlib
turtle_tf
snow@snow-ubuntu:~$
```

```
$ rospack find turtlesim
$ rospack list
$ rospack list | grep turtle
$ rospack depends-on turtlesim
$ rospack depends turtlesim
```



```
snow@snow-ubuntu:~$ rospack depends turtlesim
cpp_common
rostime
roscpp_traits
roscpp_serialization
catkin
genmsg
genpy
message_runtime
std_msgs
geometry_msgs
gencpp
geneus
gennodejs
genlisp
message_generation
rosbuild
rosconsole
rosgraph_msgs
xmlrpcpp
roscpp
ros_environment
rospack
roslib
std_srvs
snow@snow-ubuntu:~$
```

ROS Information Commands

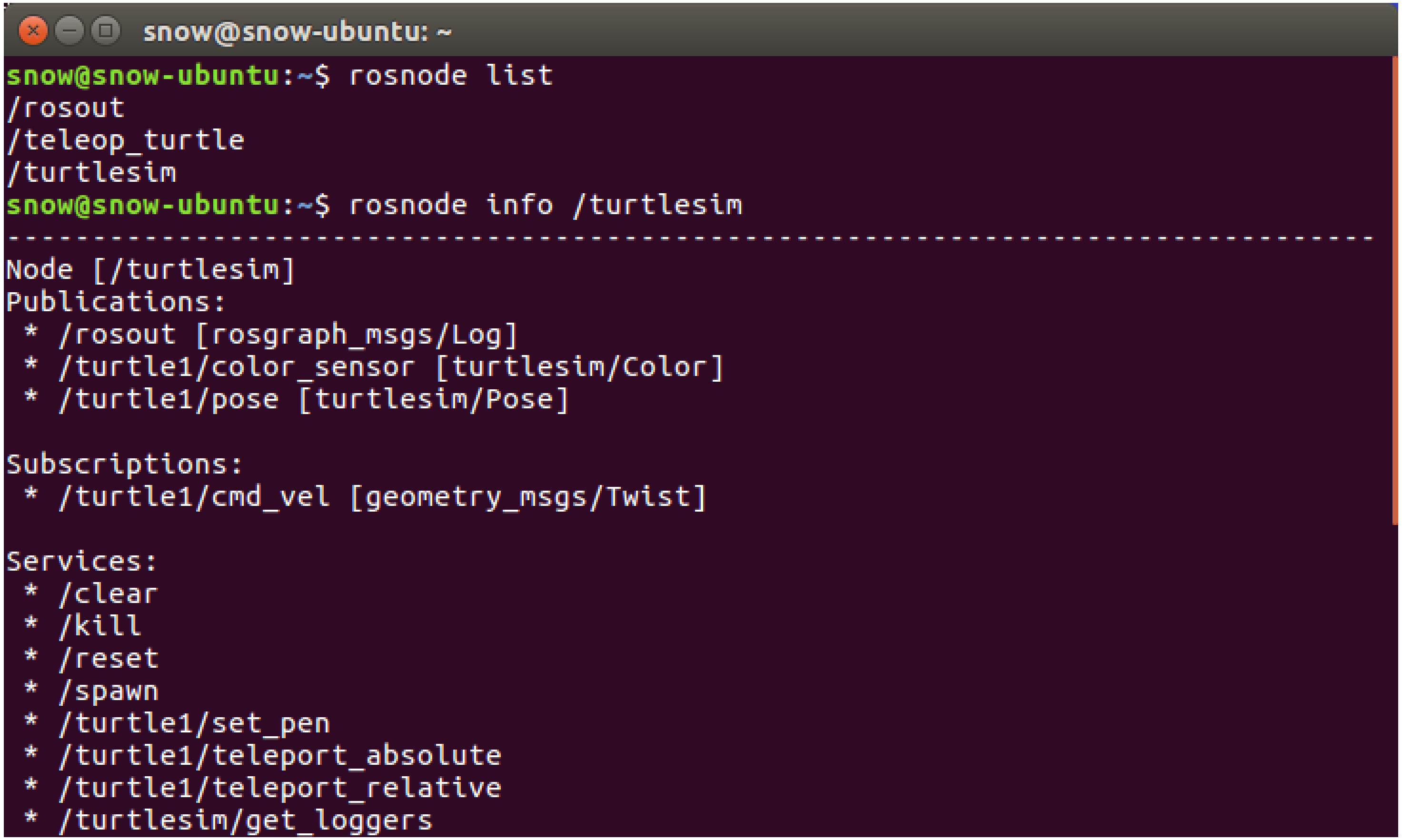
rosnode

- displays information about the ROS nodes that are currently running.
- Usage:

```
$ rosnode <command> <node_name>
```

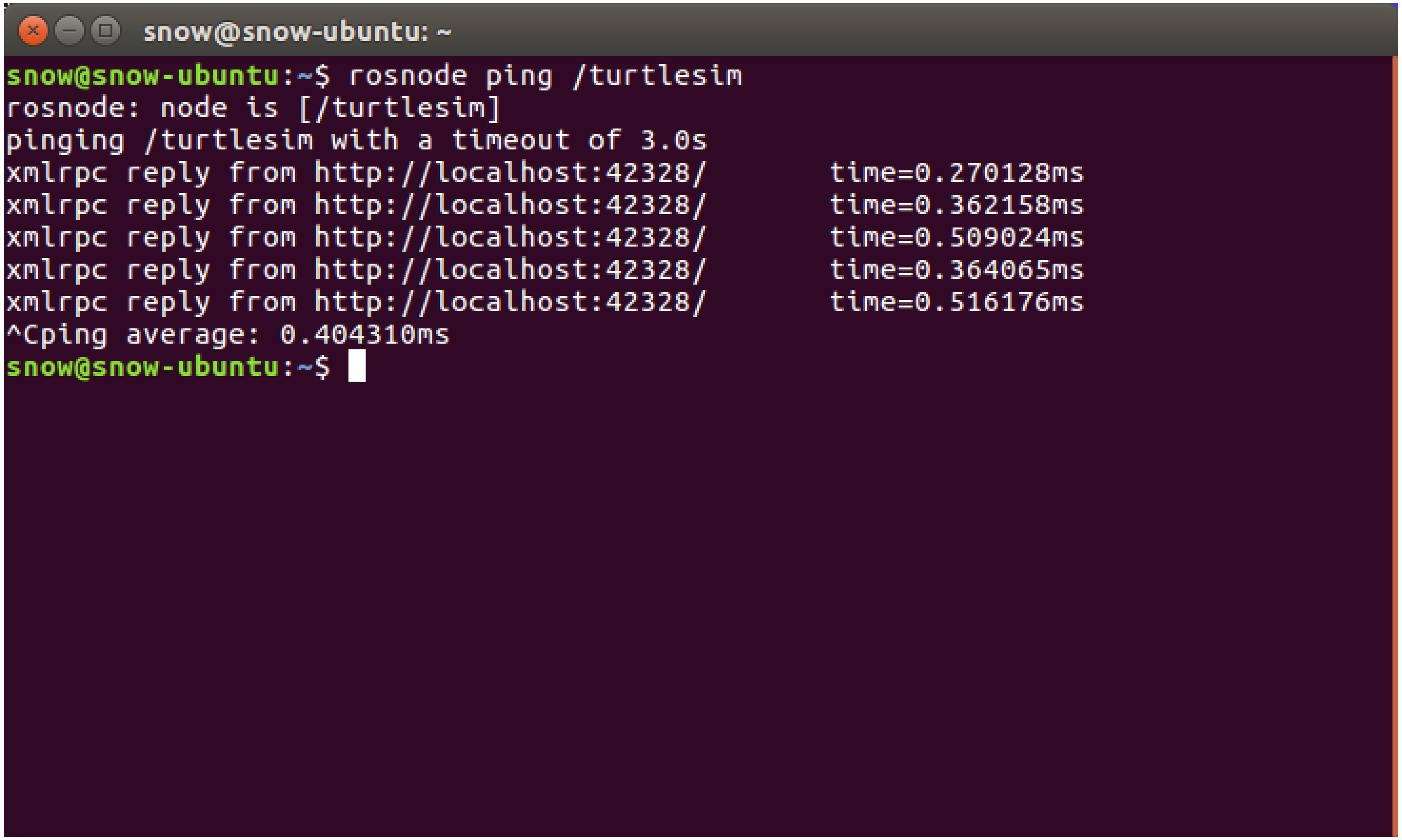
- Command
  - rosnode list # list active nodes
  - rosnode ping node_name # test connectivity to node
  - rosnode info node_name # print information about node
  - rosnode machine pc_name/ip # list nodes running on a particular machine or list machines
  - rosnode kill node_name # kill a running node
  - rosnode cleanup # purge registration information of unreachable nodes

```
$ rosnode list
$ rosnode info /turtlesim
$ rosnode ping /turtlesim
$ rosnode machine 127.0.0.1
$ rosnode kill /turtlesim
```



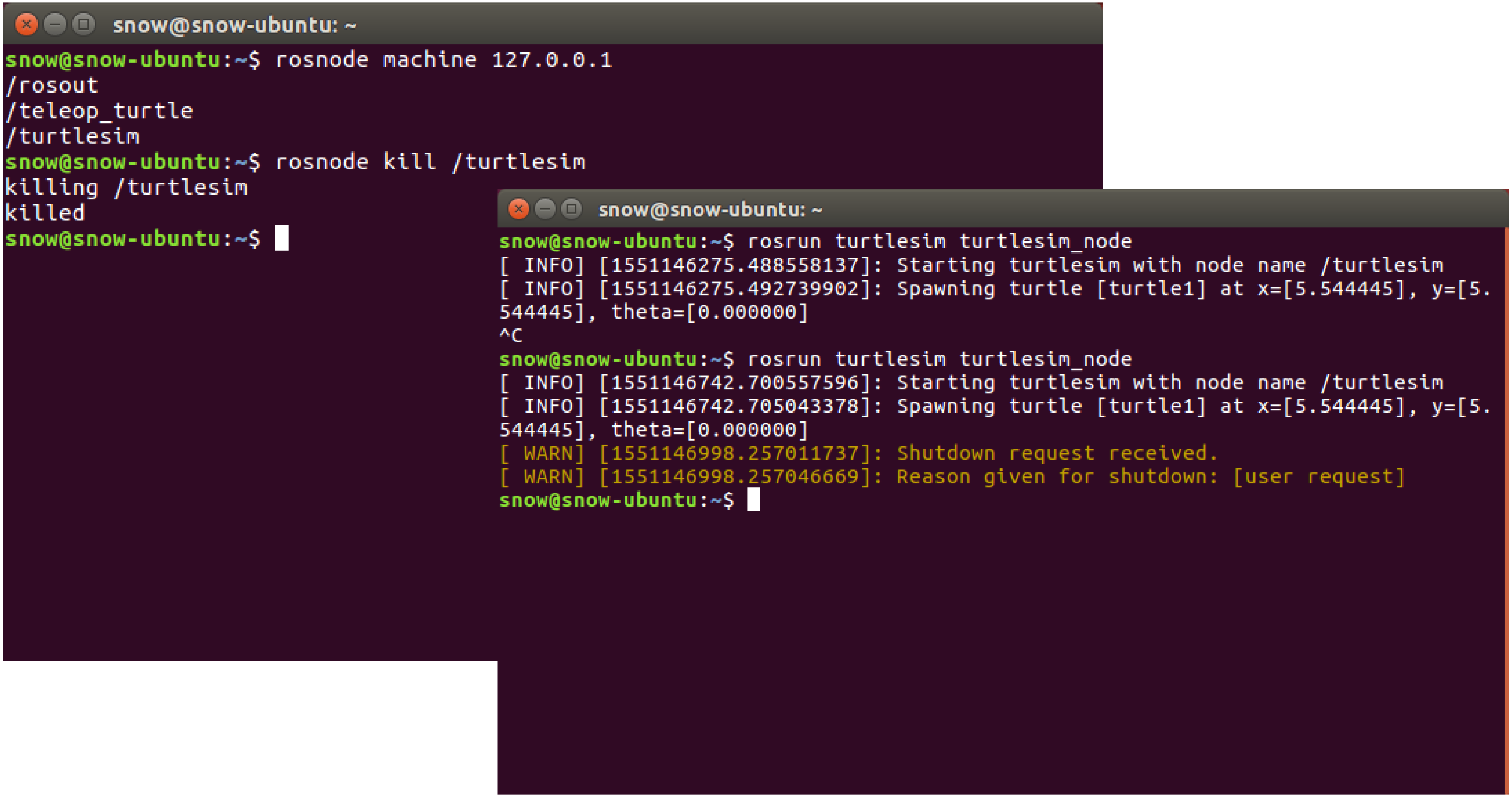
```
snow@snow-ubuntu:~$ rosnode list
/rosout
/teleop_turtle
/turtlesim
snow@snow-ubuntu:~$ rosnode info /turtlesim
--------------------------------------------------------------------------------
Node [/turtlesim]
Publications:
 * /rosout [rosgraph_msgs/Log]
 * /turtle1/color_sensor [turtlesim/Color]
 * /turtle1/pose [turtlesim/Pose]

Subscriptions:
 * /turtle1/cmd_vel [geometry_msgs/Twist]

Services:
 * /clear
 * /kill
 * /reset
 * /spawn
 * /turtle1/set_pen
 * /turtle1/teleport_absolute
 * /turtle1/teleport_relative
 * /turtlesim/get_loggers
```

```
$ rosnode list
$ rosnode info /turtlesim
$ rosnode ping /turtlesim
$ rosnode machine 127.0.0.1
$ rosnode kill /turtlesim
```



```
snow@snow-ubuntu:~$ rosnode ping /turtlesim
rosnode: node is [/turtlesim]
pinging /turtlesim with a timeout of 3.0s
xmlrpc reply from http://localhost:42328/       time=0.270128ms
xmlrpc reply from http://localhost:42328/       time=0.362158ms
xmlrpc reply from http://localhost:42328/       time=0.509024ms
xmlrpc reply from http://localhost:42328/       time=0.364065ms
xmlrpc reply from http://localhost:42328/       time=0.516176ms
^Cping average: 0.404310ms
snow@snow-ubuntu:~$ 
```

```
$ rosnode list
$ rosnode info /turtlesim
$ rosnode ping /turtlesim
$ rosnode machine 127.0.0.1
$ rosnode kill /turtlesim
```



```
snow@snow-ubuntu:~$ rosnode machine 127.0.0.1
/rosout
/teleop_turtle
/turtlesim
snow@snow-ubuntu:~$ rosnode kill /turtlesim
killing /turtlesim
killed
snow@snow-ubuntu:~$
```

```
snow@snow-ubuntu:~$ rosrun turtlesim turtlesim_node
[ INFO] [1551146275.488558137]: Starting turtlesim with node name /turtlesim
[ INFO] [1551146275.492739902]: Spawning turtle [turtle1] at x=[5.544445], y=[5.
544445], theta=[0.000000]
^C
snow@snow-ubuntu:~$ rosrun turtlesim turtlesim_node
[ INFO] [1551146742.700557596]: Starting turtlesim with node name /turtlesim
[ INFO] [1551146742.705043378]: Spawning turtle [turtle1] at x=[5.544445], y=[5.
544445], theta=[0.000000]
[ WARN] [1551146998.257011737]: Shutdown request received.
[ WARN] [1551146998.257046669]: Reason given for shutdown: [user request]
snow@snow-ubuntu:~$
```

ROS Information Commands

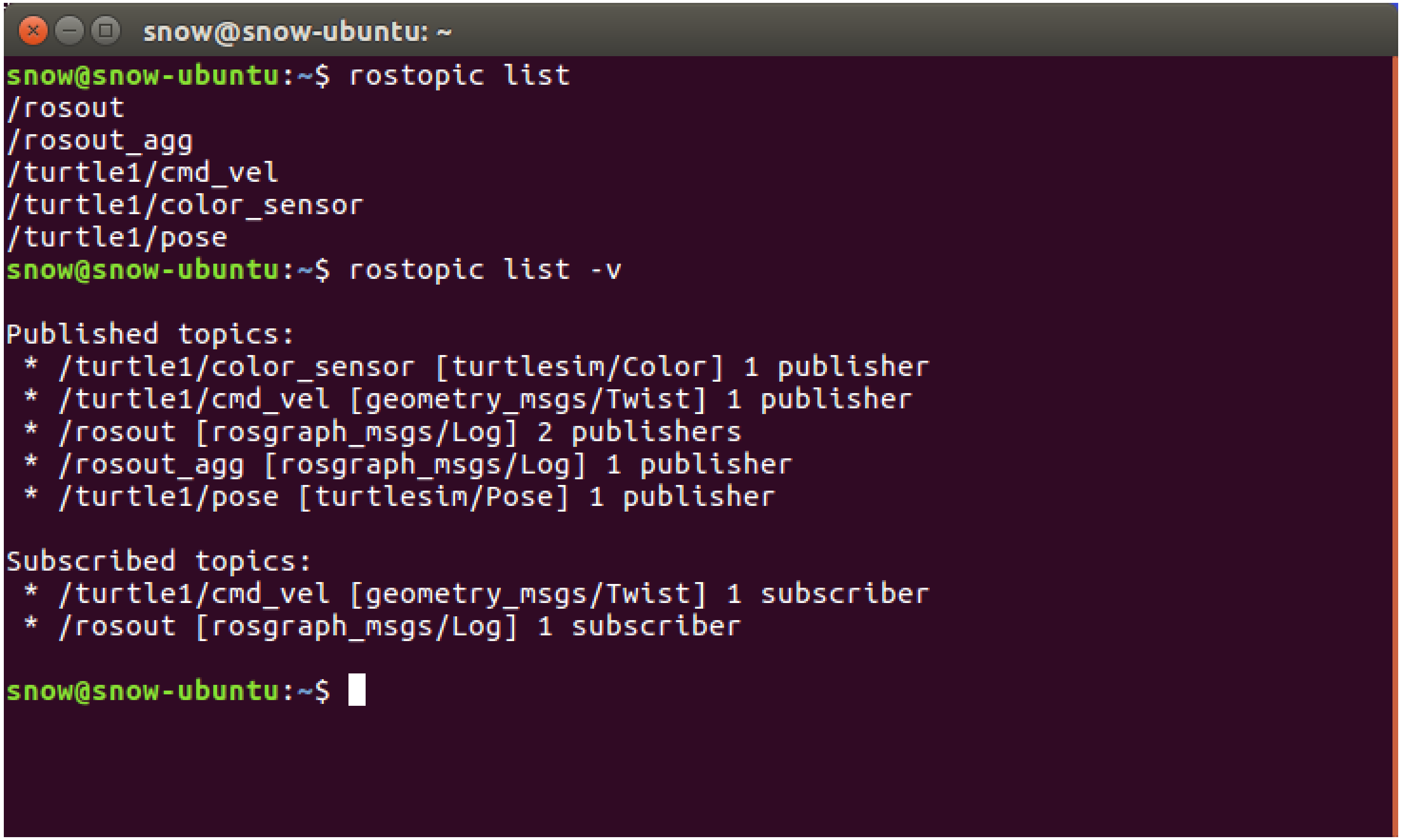
rostopic

- allows you to get information about ROS topics.
- Usage:

```
$ rostopic <command> <topic_name>
```

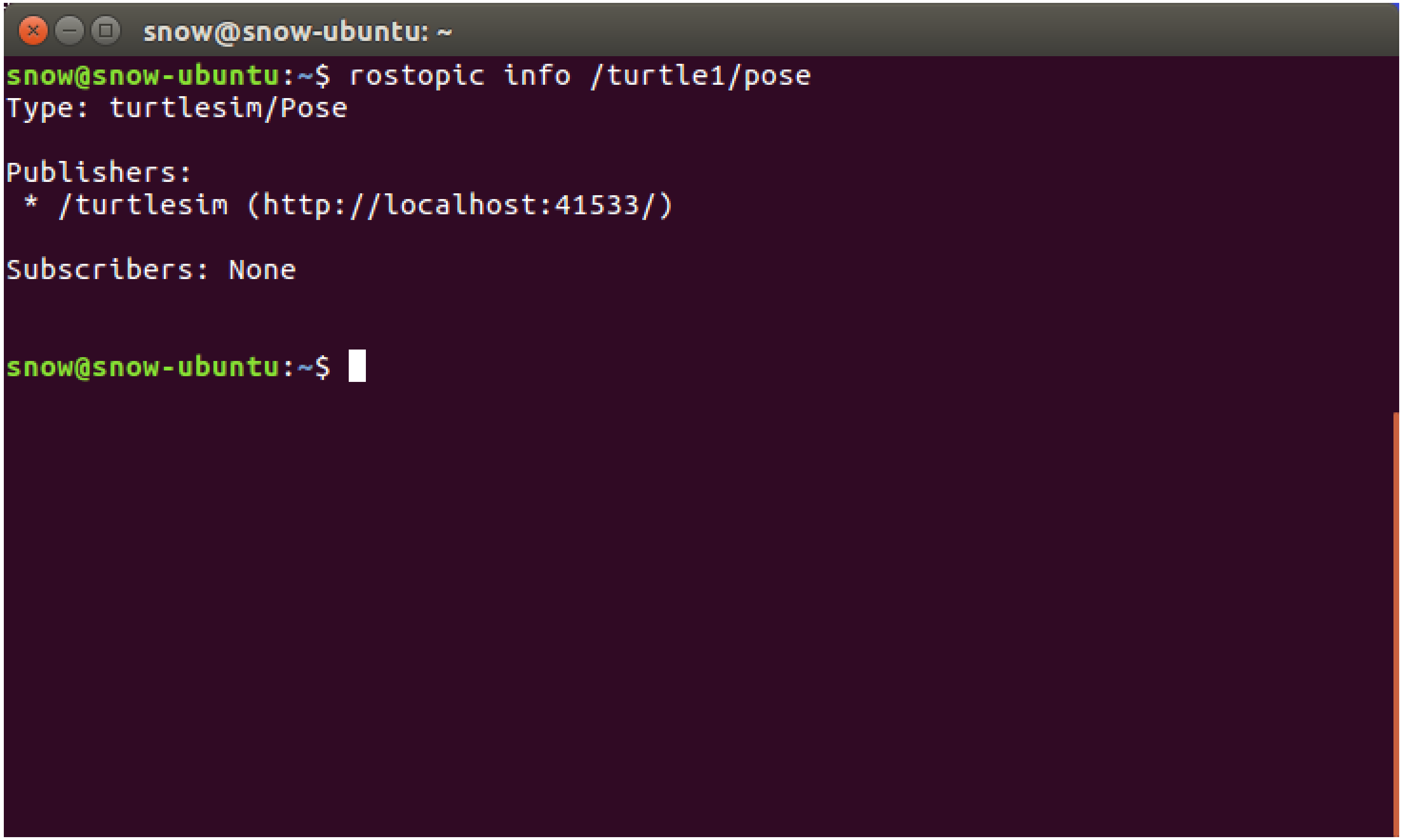
- Command
  - rostopic list # list active topics
  - rostopic bw topic_name # display bandwidth used by topic
  - rostopic echo topic_name # print messages to screen
  - rostopic find topic_name # find topics by type
  - rostopic hz topic_name # display publishing rate of topic
  - rostopic info topic_name # print information about active topic
  - rostopic type topic_name # print topic type
  - rostopic pub topic_name [msg_type] [args] # publish data to topic

```
$ rostopic list
$ rostopic list -v
```



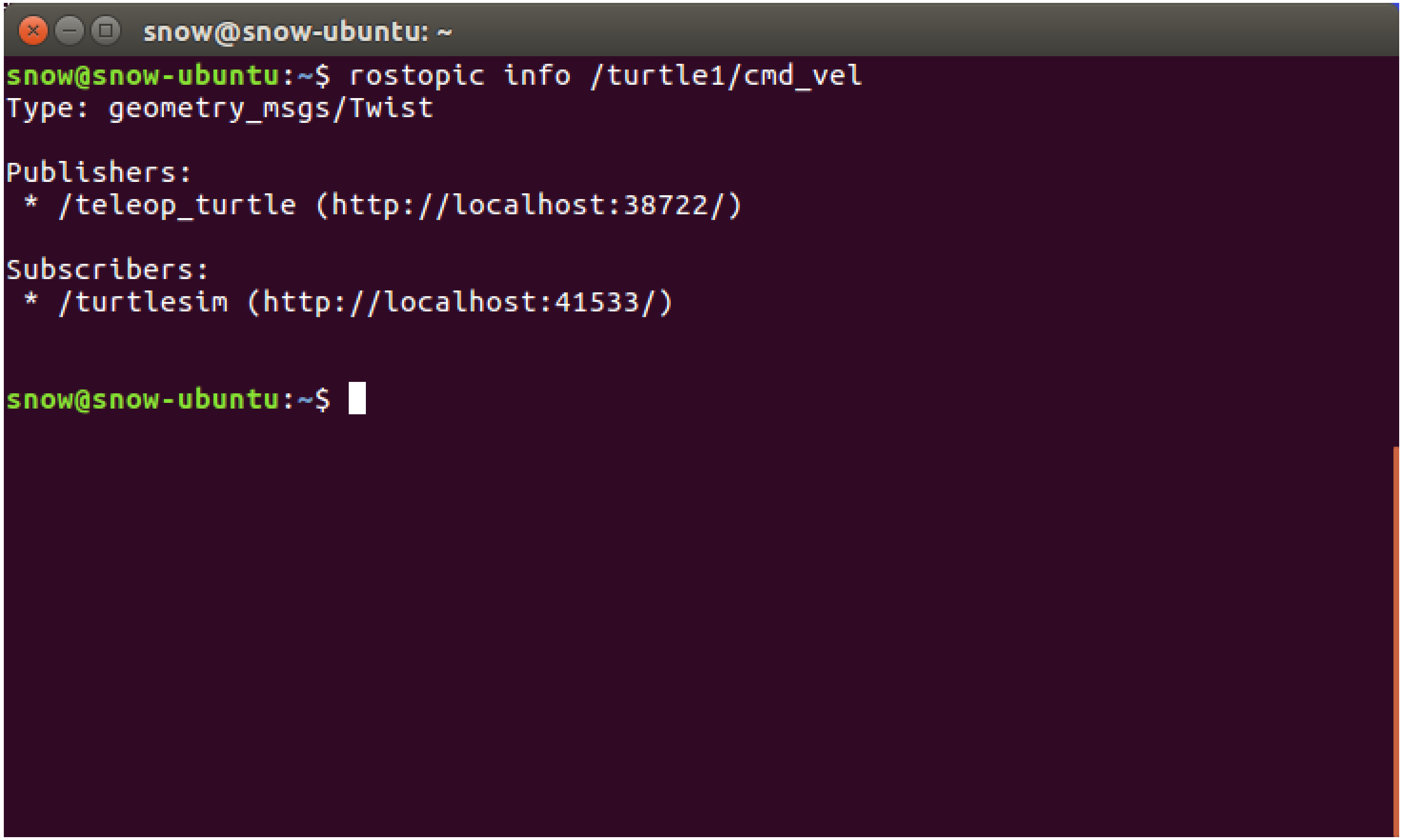
```
snow@snow-ubuntu:~$ rostopic list
/rosout
/rosout_agg
/turtle1/cmd_vel
/turtle1/color_sensor
/turtle1/pose
snow@snow-ubuntu:~$ rostopic list -v

Published topics:
 * /turtle1/color_sensor [turtlesim/Color] 1 publisher
 * /turtle1/cmd_vel [geometry_msgs/Twist] 1 publisher
 * /rosout [rosgraph_msgs/Log] 2 publishers
 * /rosout_agg [rosgraph_msgs/Log] 1 publisher
 * /turtle1/pose [turtlesim/Pose] 1 publisher

Subscribed topics:
 * /turtle1/cmd_vel [geometry_msgs/Twist] 1 subscriber
 * /rosout [rosgraph_msgs/Log] 1 subscriber

snow@snow-ubuntu:~$ 
```

```
$ rostopic info /turtle1/pose
```



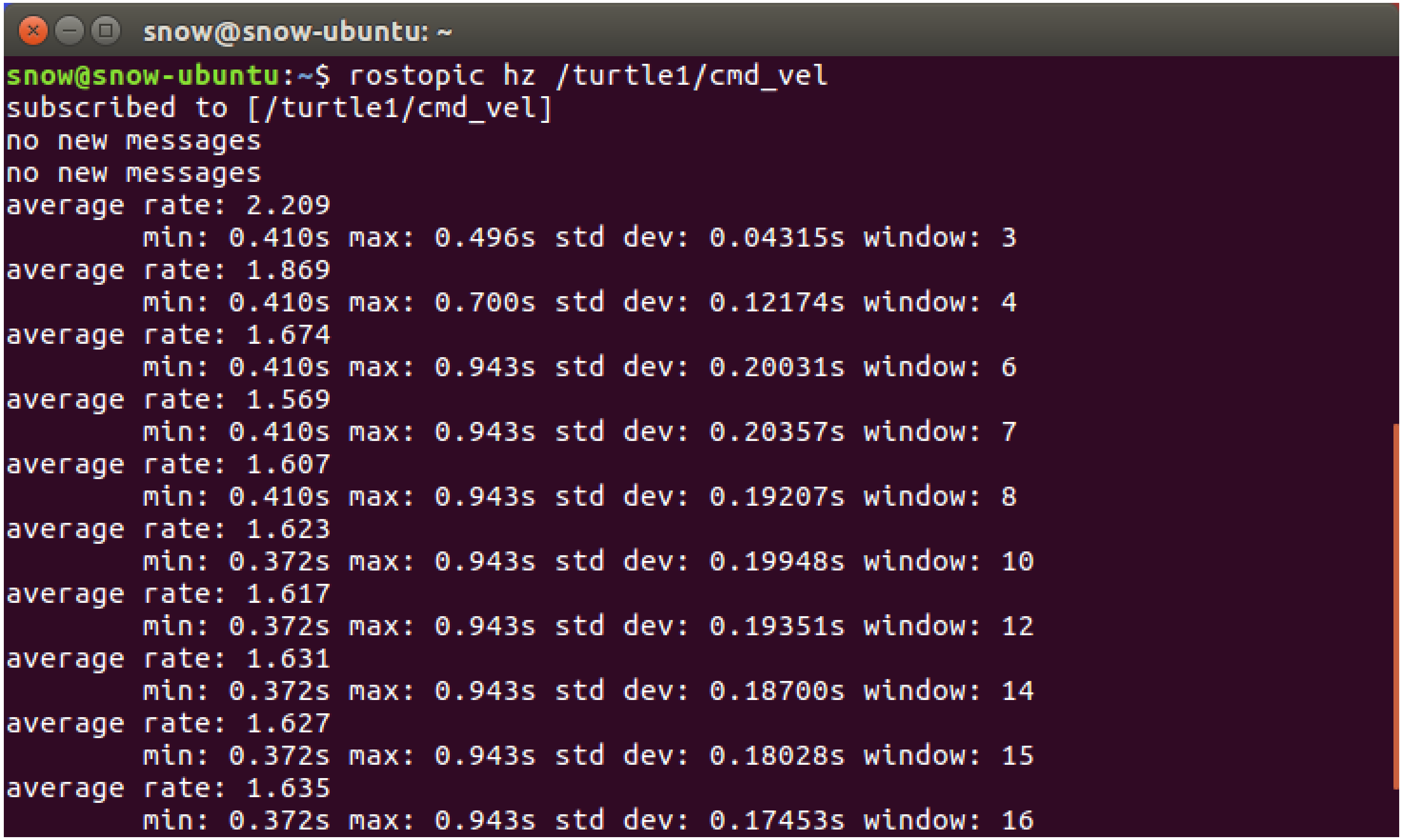
```
snow@snow-ubuntu:~$ rostopic info /turtle1/pose
Type: turtlesim/Pose

Publishers:
 * /turtlesim (http://localhost:41533/)

Subscribers: None


snow@snow-ubuntu:~$
```

```
$ rostopic info /turtle1/cmd_vel
```



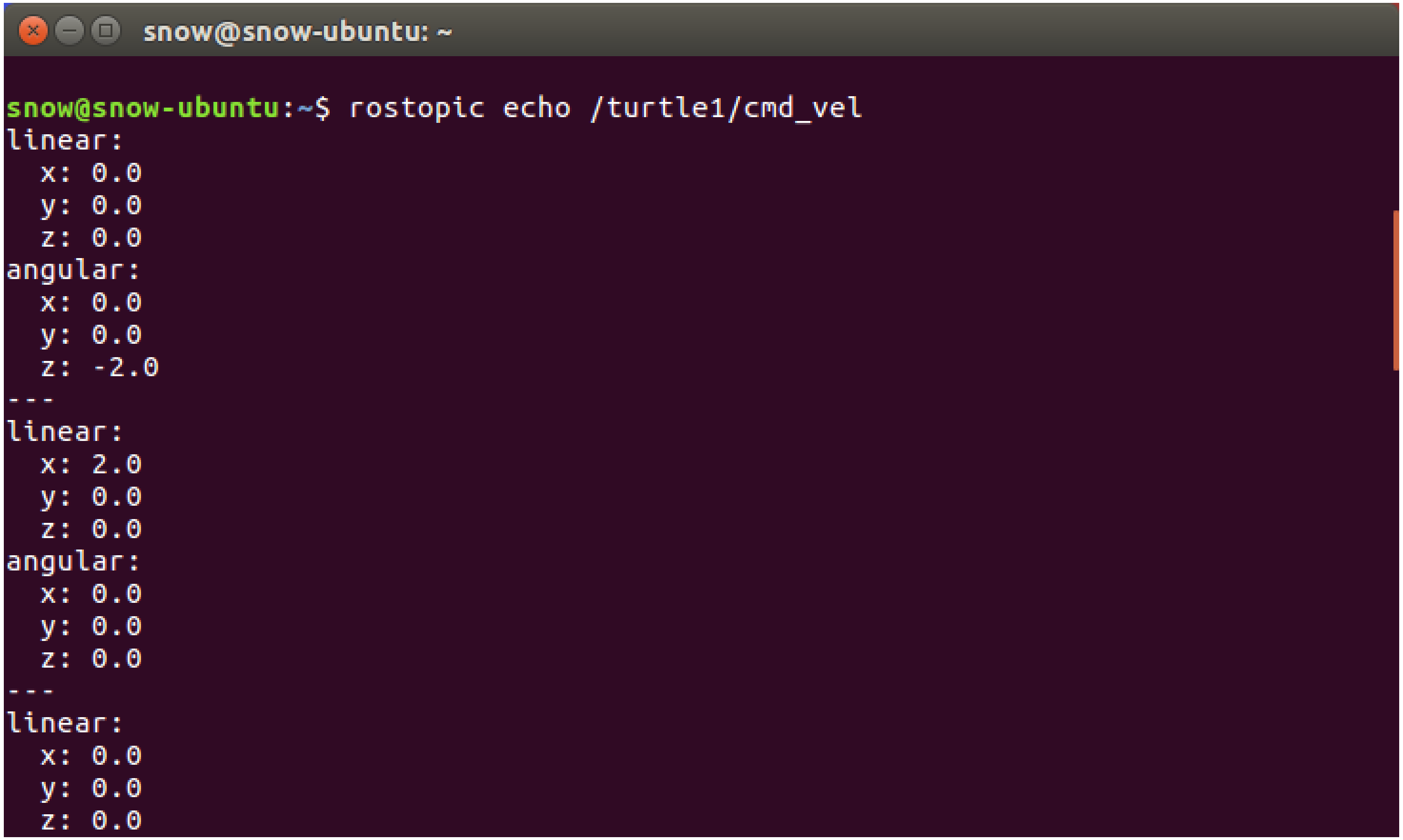
```
snow@snow-ubuntu:~$ rostopic info /turtle1/cmd_vel
Type: geometry_msgs/Twist

Publishers:
 * /teleop_turtle (http://localhost:38722/)

Subscribers:
 * /turtlesim (http://localhost:41533/)


snow@snow-ubuntu:~$ 
```

```
$ rostopic hz /turtle1/cmd_vel
```



```
snow@snow-ubuntu:~$ rostopic hz /turtle1/cmd_vel
subscribed to [/turtle1/cmd_vel]
no new messages
no new messages
average rate: 2.209
        min: 0.410s max: 0.496s std dev: 0.04315s window: 3
average rate: 1.869
        min: 0.410s max: 0.700s std dev: 0.12174s window: 4
average rate: 1.674
        min: 0.410s max: 0.943s std dev: 0.20031s window: 6
average rate: 1.569
        min: 0.410s max: 0.943s std dev: 0.20357s window: 7
average rate: 1.607
        min: 0.410s max: 0.943s std dev: 0.19207s window: 8
average rate: 1.623
        min: 0.372s max: 0.943s std dev: 0.19948s window: 10
average rate: 1.617
        min: 0.372s max: 0.943s std dev: 0.19351s window: 12
average rate: 1.631
        min: 0.372s max: 0.943s std dev: 0.18700s window: 14
average rate: 1.627
        min: 0.372s max: 0.943s std dev: 0.18028s window: 15
average rate: 1.635
        min: 0.372s max: 0.943s std dev: 0.17453s window: 16
```

```
$ rostopic echo /turtle1/cmd_vel
```



```
snow@snow-ubuntu:~$ rostopic echo /turtle1/cmd_vel
linear:
  x: 0.0
  y: 0.0
  z: 0.0
angular:
  x: 0.0
  y: 0.0
  z: -2.0
---
linear:
  x: 2.0
  y: 0.0
  z: 0.0
angular:
  x: 0.0
  y: 0.0
  z: 0.0
---
linear:
  x: 0.0
  y: 0.0
  z: 0.0
```

◆ rostopic pub

- publishes data on to a topic currently advertised
- Usage:

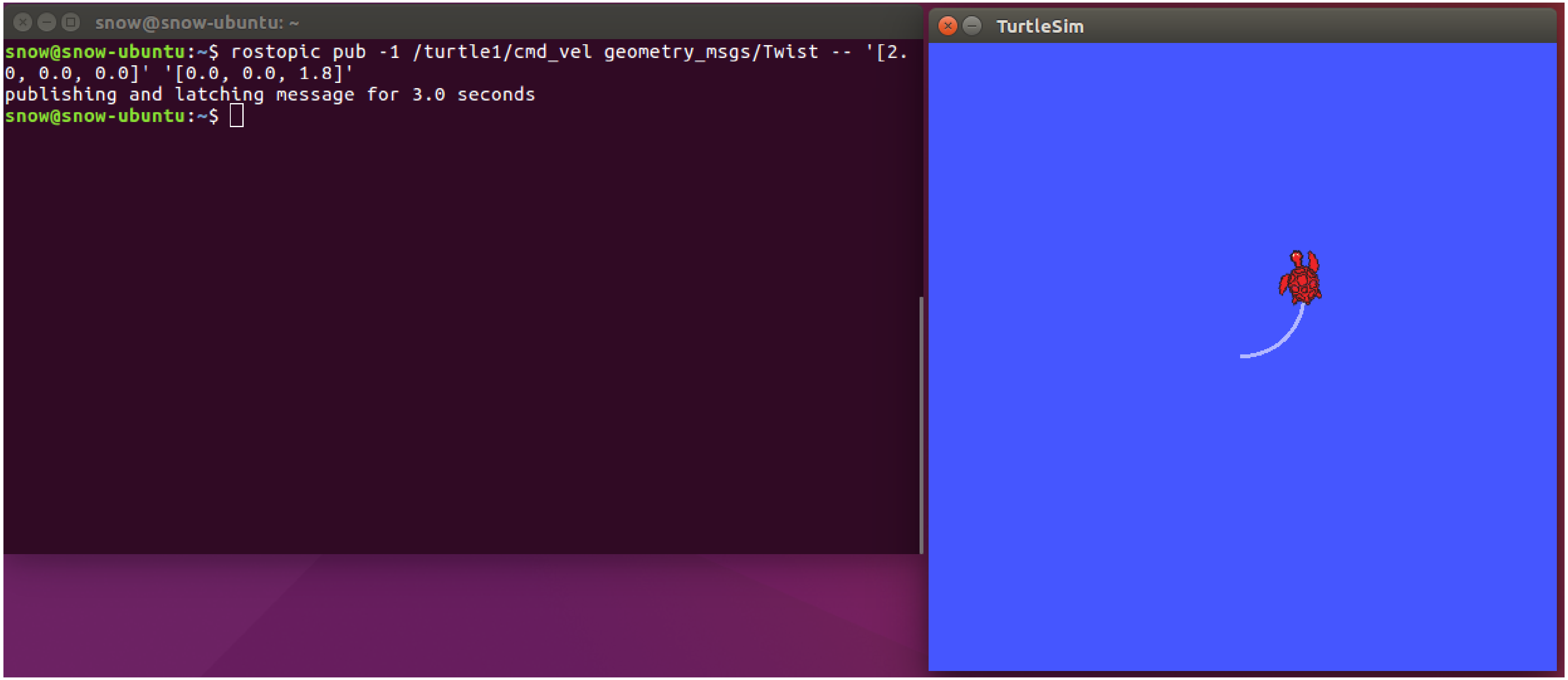
```
$ rostopic pub [topic_name] [msg_type] [args]
```

- Try

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

  - '-1' : publish once
  - '/turtle1/cmd_vel' : topic name
  - 'geometry_msgs/Twist' : message type
    two vectors of three floating point elements each: *linear* and *angular*
  - '--' : the option parser that none of the following arguments is an option
  - '[2.0, 0.0, 0.0]' '[0.0, 0.0, 1.8]' :
    move at a speed of 2.0m/s in x-axis, rotate 1.8rad/sec around z-axis

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



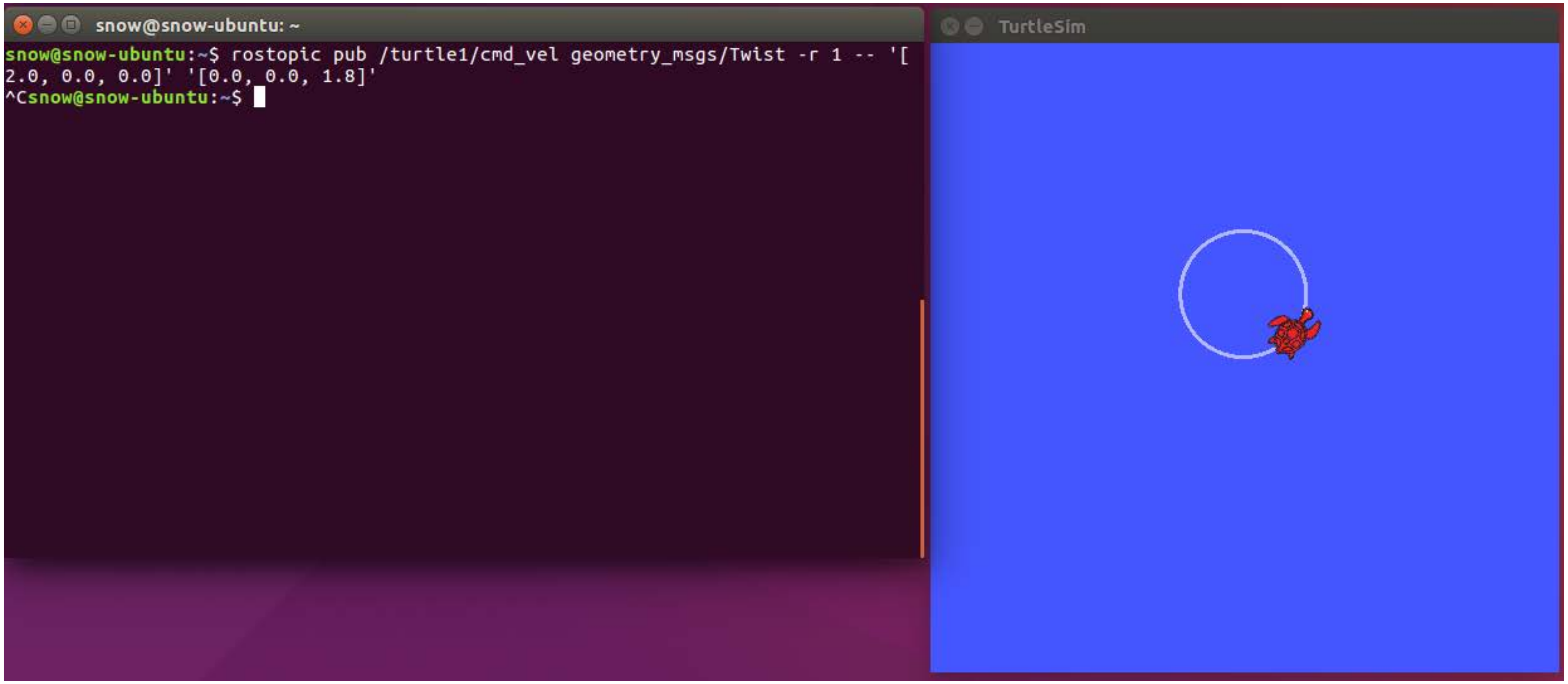
```
snow@snow-ubuntu:~$ rostopic pub -1 /turtle1/cmd_vel geometry_msgs/Twist -- '[2.
0, 0.0, 0.0]' '[0.0, 0.0, 1.8]'
publishing and latching message for 3.0 seconds
snow@snow-ubuntu:~$
```

rostopic pub

- Try

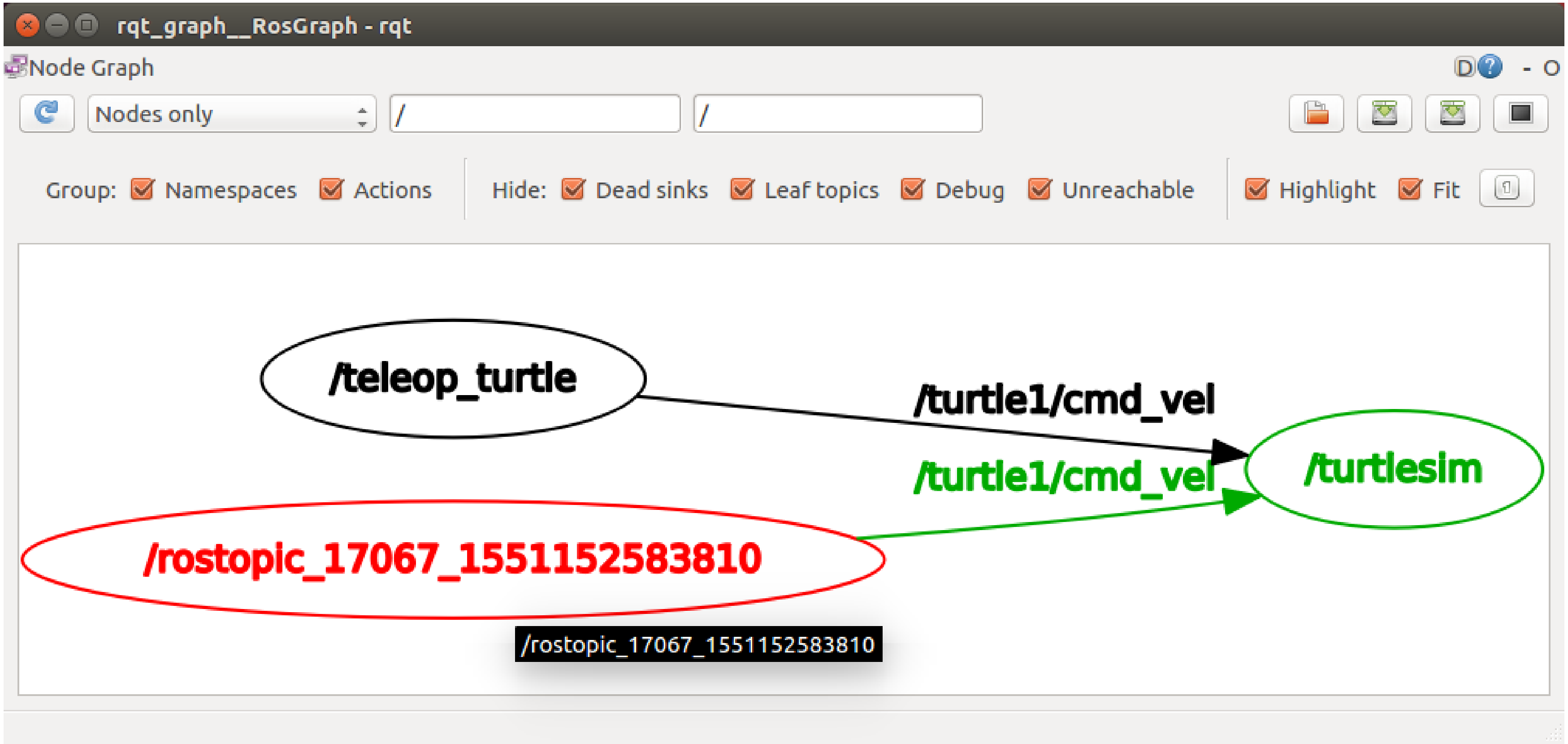
```
$ rostopic pub /turtle1/cmd_vel geometry_msgs/Twist -r 1 --
'[2.0, 0.0, 0.0]' '[0.0, 0.0, 1.8]'
```

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



```
snow@snow-ubuntu:~$ rostopic pub /turtle1/cmd_vel geometry_msgs/Twist -r 1 -- '[
2.0, 0.0, 0.0]' '[0.0, 0.0, 1.8]'
^Csnow@snow-ubuntu:~$
```

- # rqt_graph



rqt_graph__RosGraph - rqt
Node Graph
Nodes only
/
/
Group: ☑ Namespaces ☑ Actions
Hide: ☑ Dead sinks ☑ Leaf topics ☑ Debug ☑ Unreachable
☑ Highlight ☑ Fit
/teleop_turtle
/turtle1/cmd_vel
/turtle1/cmd_vel
/turtlesim
/rostopic_17067_1551152583810
/rostopic_17067_1551152583810

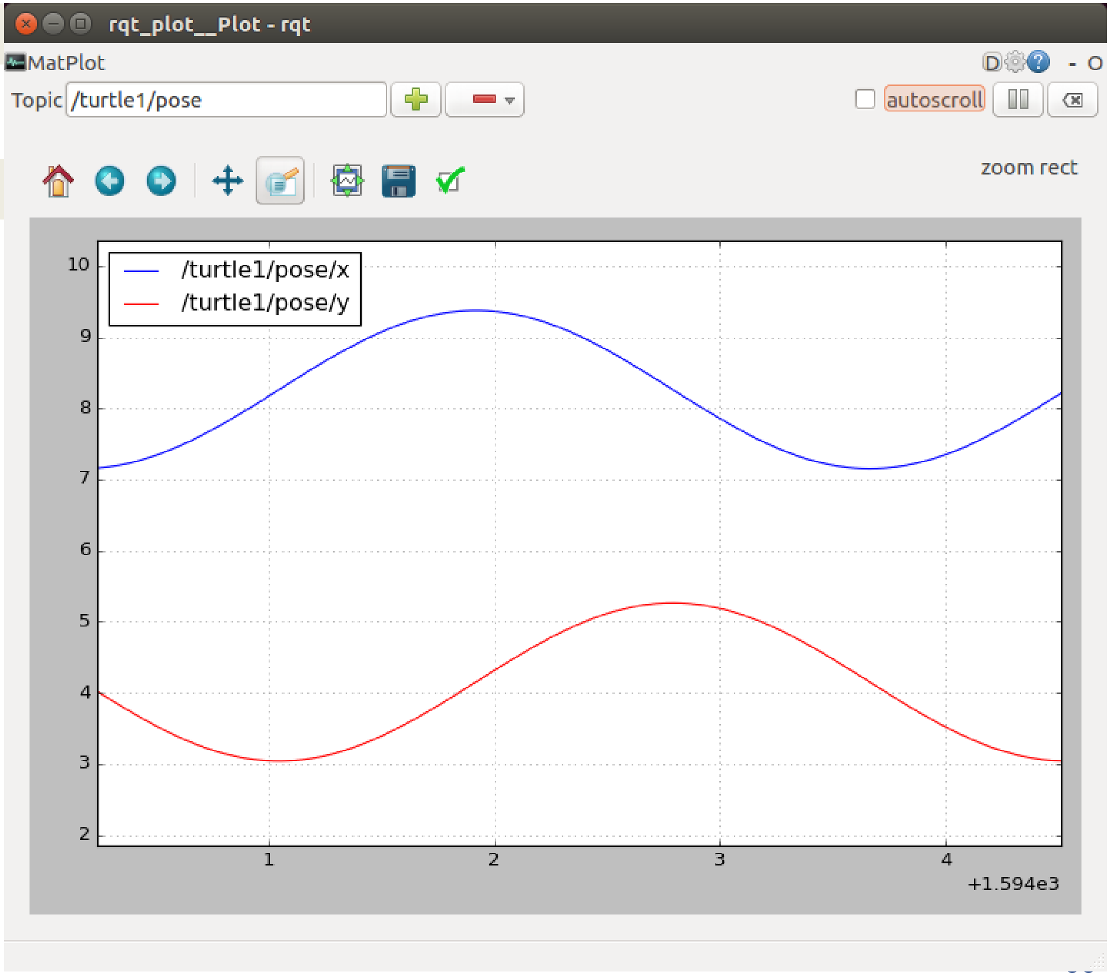
rqt_plot

- displays a scrolling time plot of the data published on topics.
- Try

```
$ rqt_plot
```

or

```
$ rqt_plot /turtle1/pose
```



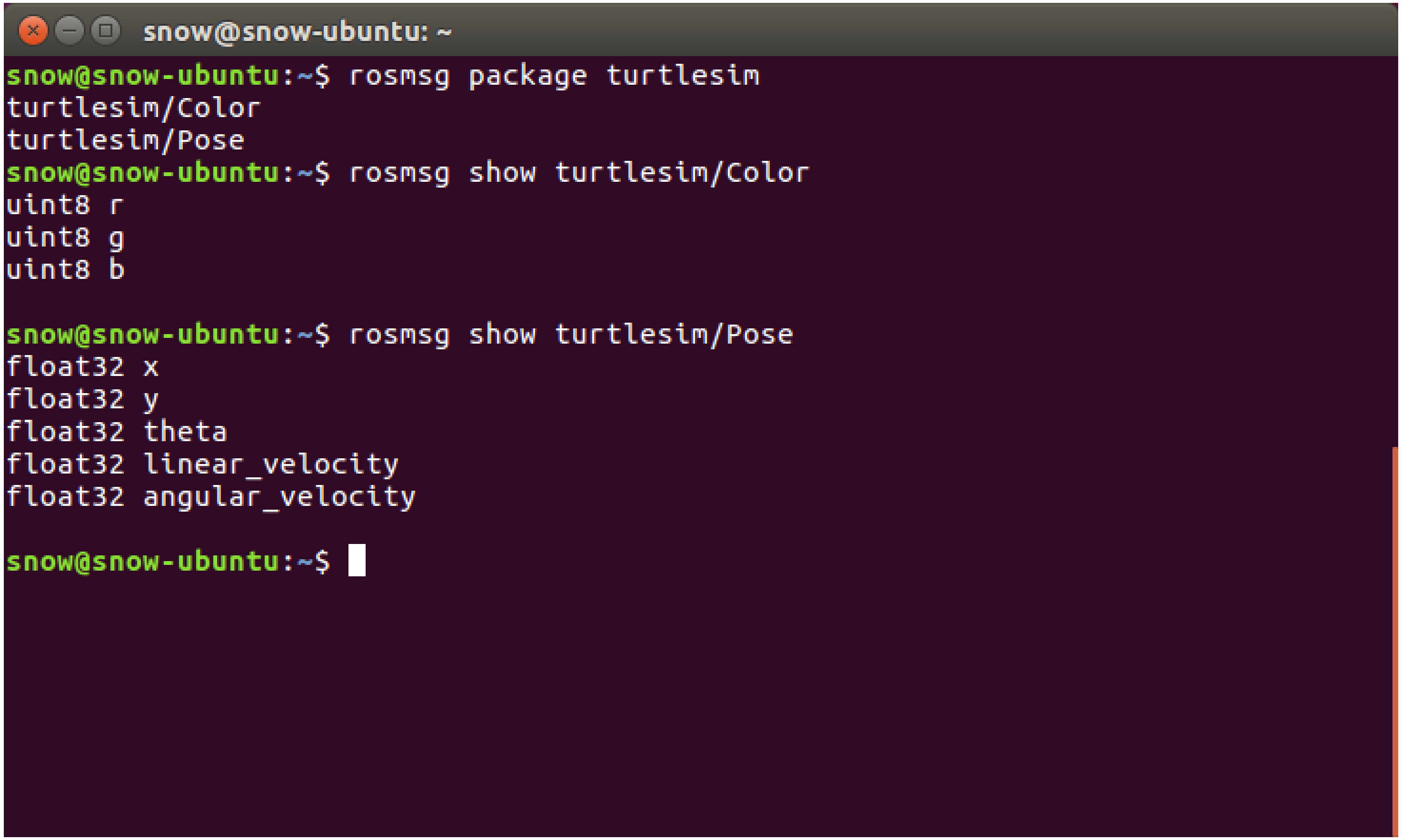
Topic /turtle1/pose

□ autoscroll

ROS Information Commands

- **rosmsg**
  - a command-line tool for displaying infor-mation about ROS Message types
  - Usage: `$ rosmsg <command> <msg_name>`
  - Command
    - rosmsg list # List all messages
    - rosmsg show msg_name # Show message description
    - rosmsg md5 msg_name # Display message md5sum
    - rosmsg package pkg_name # List messages in a package
    - rosmsg packages # List packages that contain messages

```
$ rosmsg list
$ rosmsg package turtlesim
$ rosmsg show turtlesim/Pose
$ rosmsg show turtlesim/Color
```



```
snow@snow-ubuntu:~$ rosmsg package turtlesim
turtlesim/Color
turtlesim/Pose
snow@snow-ubuntu:~$ rosmsg show turtlesim/Color
uint8 r
uint8 g
uint8 b

snow@snow-ubuntu:~$ rosmsg show turtlesim/Pose
float32 x
float32 y
float32 theta
float32 linear_velocity
float32 angular_velocity

snow@snow-ubuntu:~$ 
```

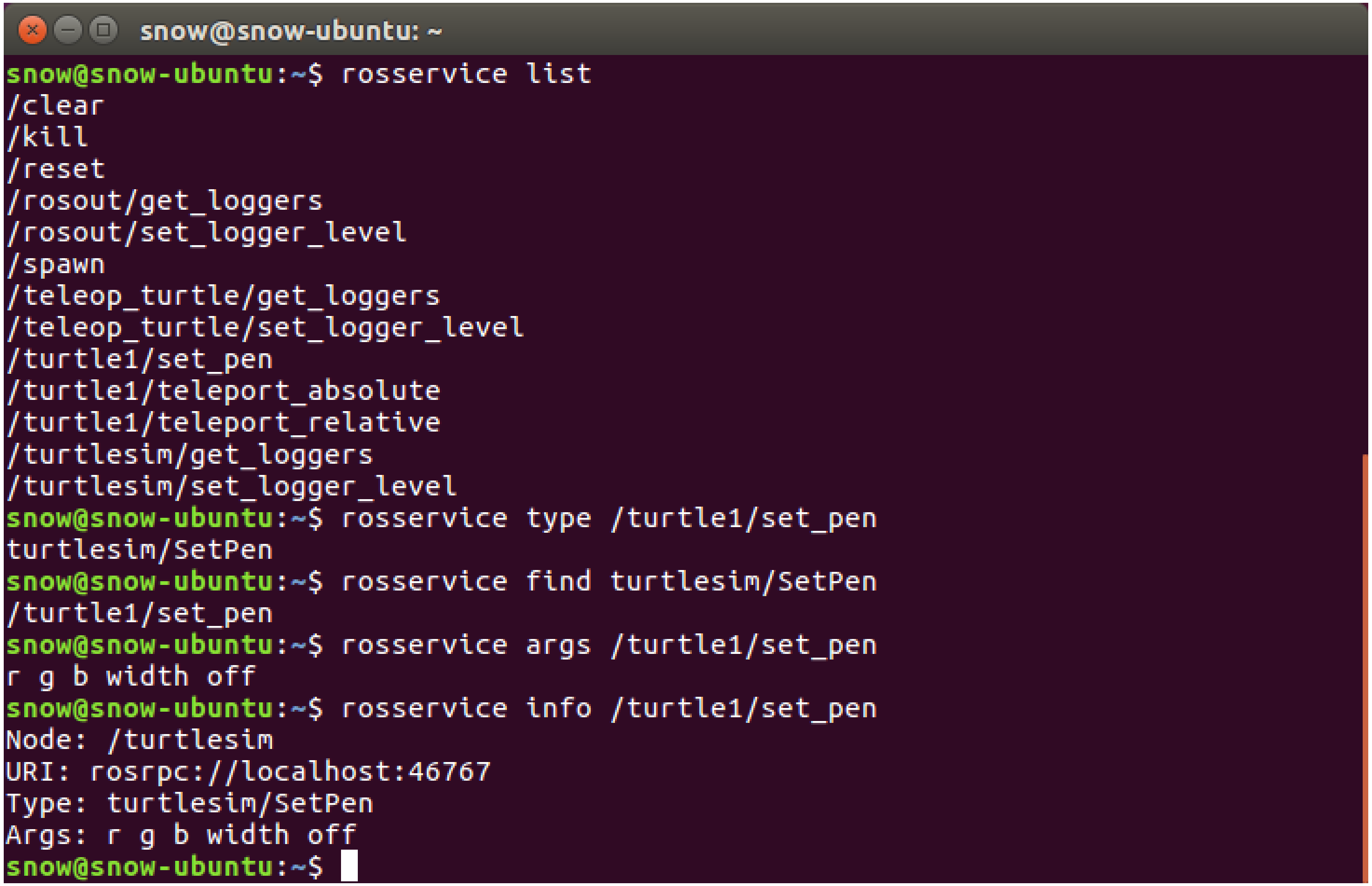
ROS Information Commands

- **rosservice**
  - Command-line tool for finding or calling service messages from the ROS Master
  - Usage:

```
$ rosservice <command> <service_name>
```

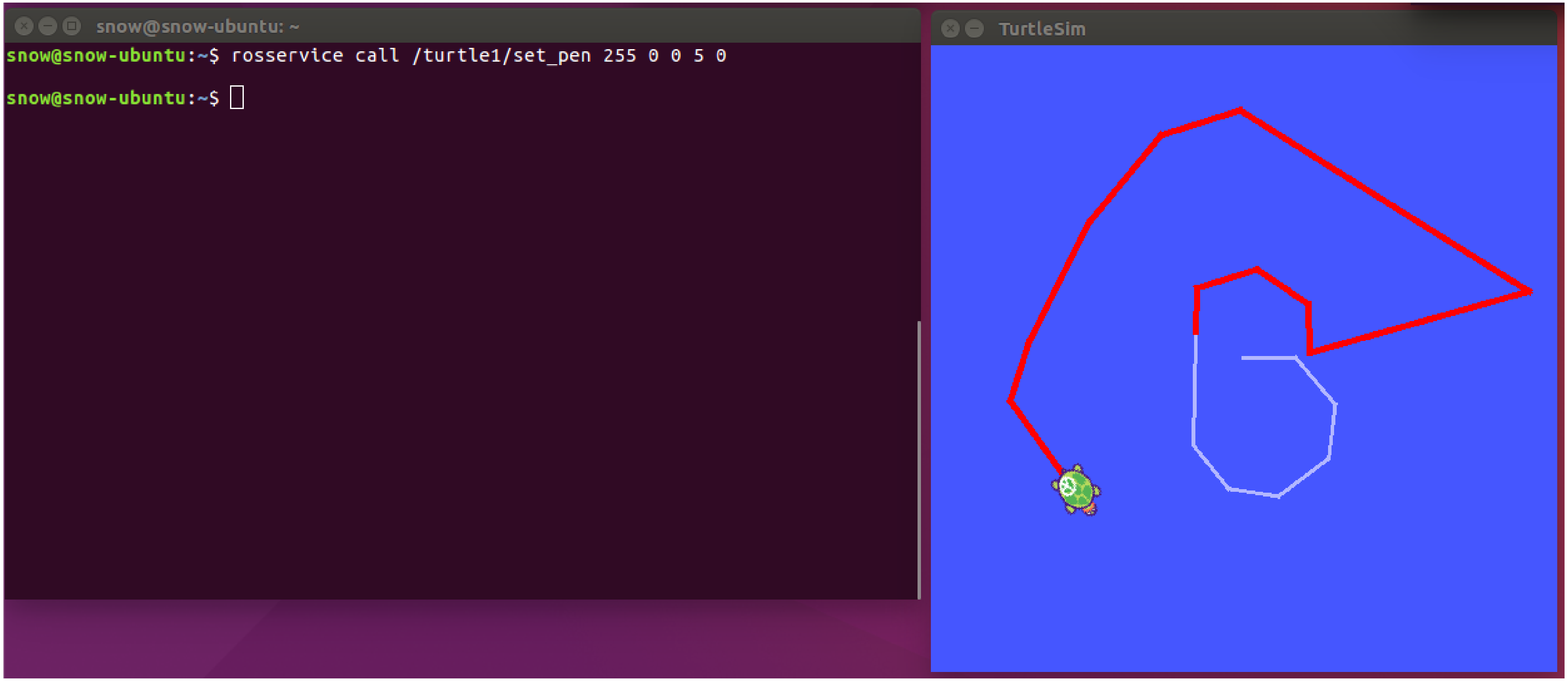
  - Command
    - rosservice list # list active services
    - rosservice args service_name # print service arguments
    - rosservice find service_name # find services by service type
    - rosservice info service_name # print information about service
    - rosservice type service_name # print service type
    - rosservice uri service_name # print service ROSRPC uri
    - rosservice call service_name param # call the service with the provided args

```
$ rosservice list
$ rosservice type /turtle1/set_pen
$ rosservice find turtlesim/SetPen
$ rosservice args /turtle1/set_pen
$ rosservice info/turtle1/set_pen
```



```
snow@snow-ubuntu:~$ rosservice list
/clear
/kill
/reset
/rosout/get_loggers
/rosout/set_logger_level
/spawn
/teleop_turtle/get_loggers
/teleop_turtle/set_logger_level
/turtle1/set_pen
/turtle1/teleport_absolute
/turtle1/teleport_relative
/turtlesim/get_loggers
/turtlesim/set_logger_level
snow@snow-ubuntu:~$ rosservice type /turtle1/set_pen
turtlesim/SetPen
snow@snow-ubuntu:~$ rosservice find turtlesim/SetPen
/turtle1/set_pen
snow@snow-ubuntu:~$ rosservice args /turtle1/set_pen
r g b width off
snow@snow-ubuntu:~$ rosservice info /turtle1/set_pen
Node: /turtlesim
URI: rosrpc://localhost:46767
Type: turtlesim/SetPen
Args: r g b width off
snow@snow-ubuntu:~$
```

```
$ rosservice call /turtle1/set_pen 255 0 0 5 0
```



```
snow@snow-ubuntu:~$ rosservice call /turtle1/set_pen 255 0 0 5 0

snow@snow-ubuntu:~$
```

ROS Information Commands

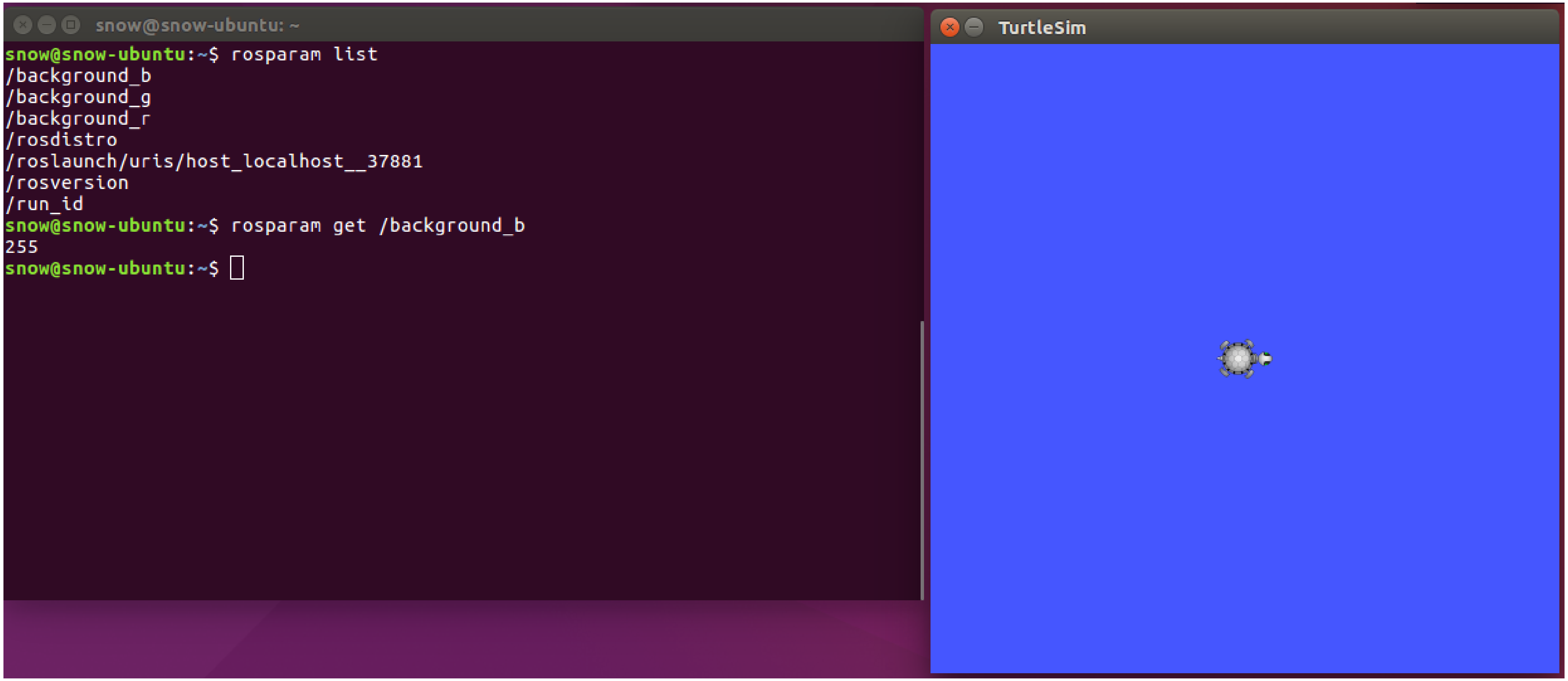
rosparam

- Command-line tool for getting, setting, and deleting parameters from the ROS Parameter Server
- Usage:

```
$ rosparam <command> <param_name>
```

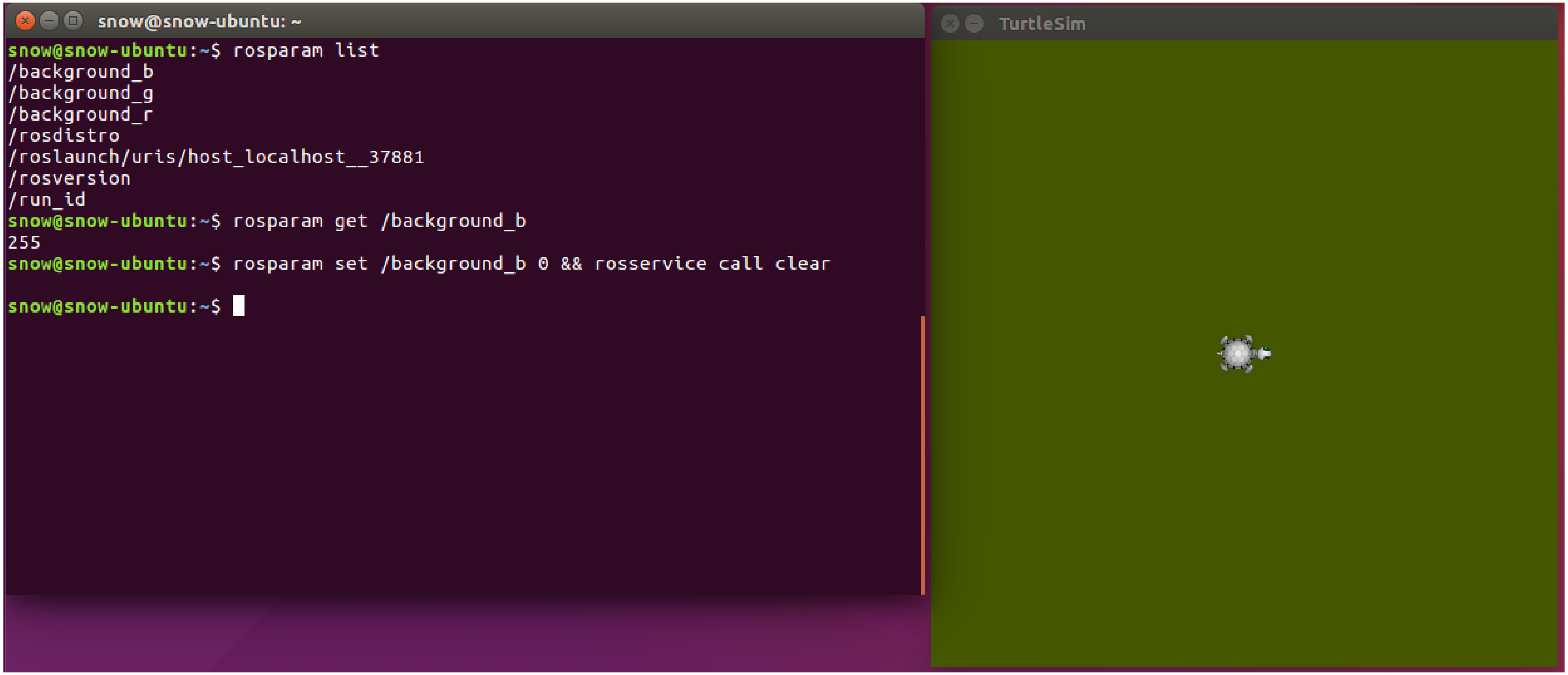
- Command
  - rosparam list # list parameter names
  - rosparam set param_name # set parameter
  - rosparam get param_name # get parameter
  - rosparam load file_name # load parameters from file
  - rosparam dump file_name # dump parameters to file
  - rosparam delete param_name # delete parameter

```
$ rosparam list
$ rosparam get /background_b
```



```
snow@snow-ubuntu:~$ rosparam list
/background_b
/background_g
/background_r
/rosdistro
/roslaunch/uris/host_localhost__37881
/rosversion
/run_id
snow@snow-ubuntu:~$ rosparam get /background_b
255
snow@snow-ubuntu:~$ 
```

```
$ rosparam set /background_b 0 && rosservice call clear
$ rosparam delete /background_b
```



```
snow@snow-ubuntu:~$ rosparam list
/background_b
/background_g
/background_r
/rosdistro
/roslaunch/uris/host_localhost__37881
/rosversion
/run_id
snow@snow-ubuntu:~$ rosparam get /background_b
255
snow@snow-ubuntu:~$ rosparam set /background_b 0 && rosservice call clear

snow@snow-ubuntu:~$ 
```

ROS Simple Programming

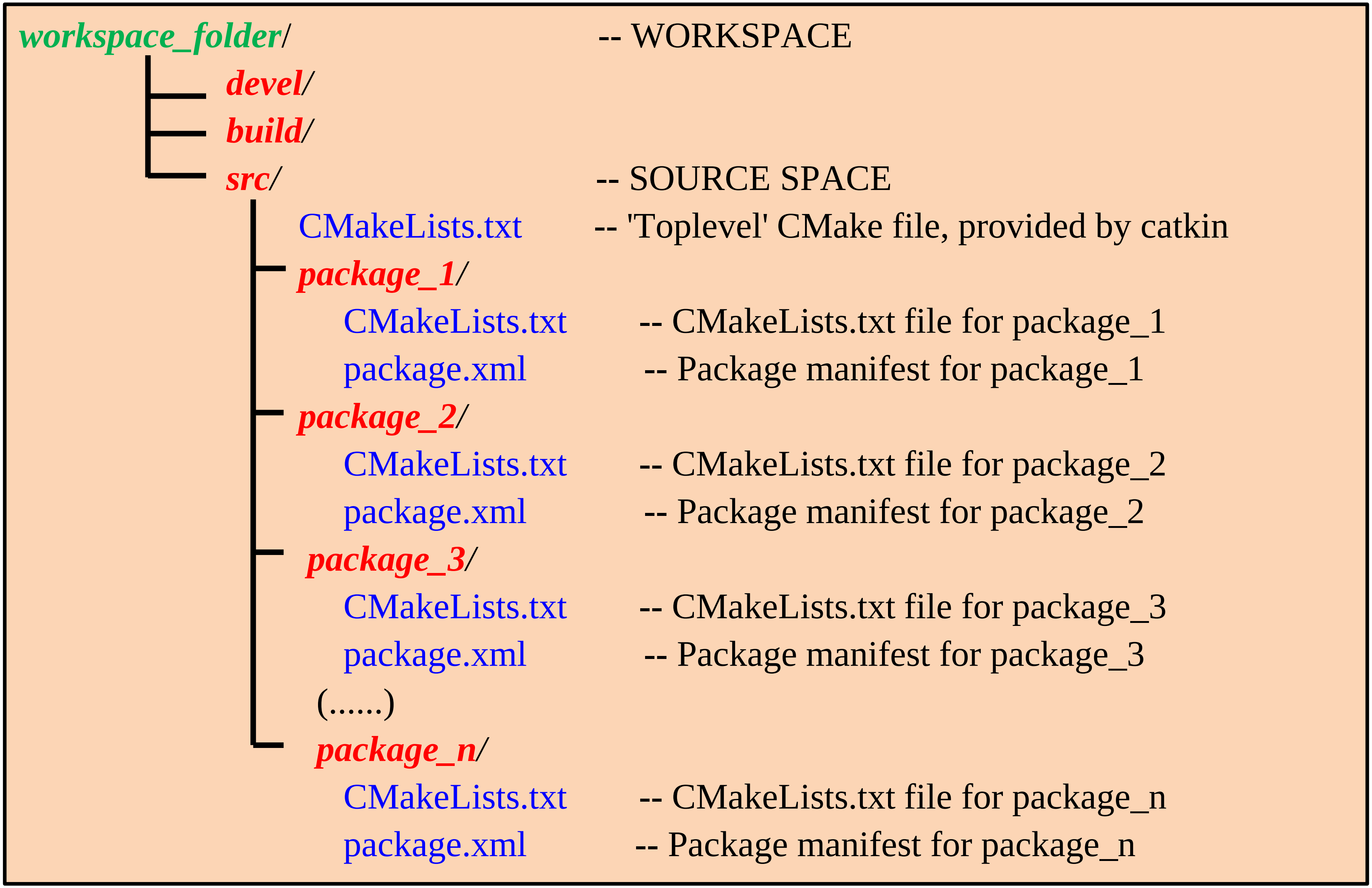
ROS File System

Catkin Workspace

- A workspace in which one or more catkin packages
- Contains up to four different spaces:

Space	Descriptions
Source space	Contains the source code of catkin packages. Each folder within the source space contains one or more catkin packages.
Build Space	is where CMake is invoked to build the catkin packages in the source space. CMake and catkin keep their cache information and other intermediate files here.
Development Space	is where built targets are placed prior to being installed
Install Space	Once targets are built, they can be installed into the install space by invoking the install target.

ROS File System



```
workspace_folder/                  -- WORKSPACE
        devel/
        build/
        src/                       -- SOURCE SPACE
            CMakeLists.txt         -- 'Toplevel' CMake file, provided by catkin
            package_1/
              CMakeLists.txt         -- CMakeLists.txt file for package_1
              package.xml            -- Package manifest for package_1
            package_2/
              CMakeLists.txt         -- CMakeLists.txt file for package_2
              package.xml            -- Package manifest for package_2
            package_3/
              CMakeLists.txt         -- CMakeLists.txt file for package_3
              package.xml            -- Package manifest for package_3
             (......)
            package_n/
              CMakeLists.txt         -- CMakeLists.txt file for package_n
              package.xml            -- Package manifest for package_n
```

Catkin Workspace

ROS Installation

- create a catkin workspace:

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

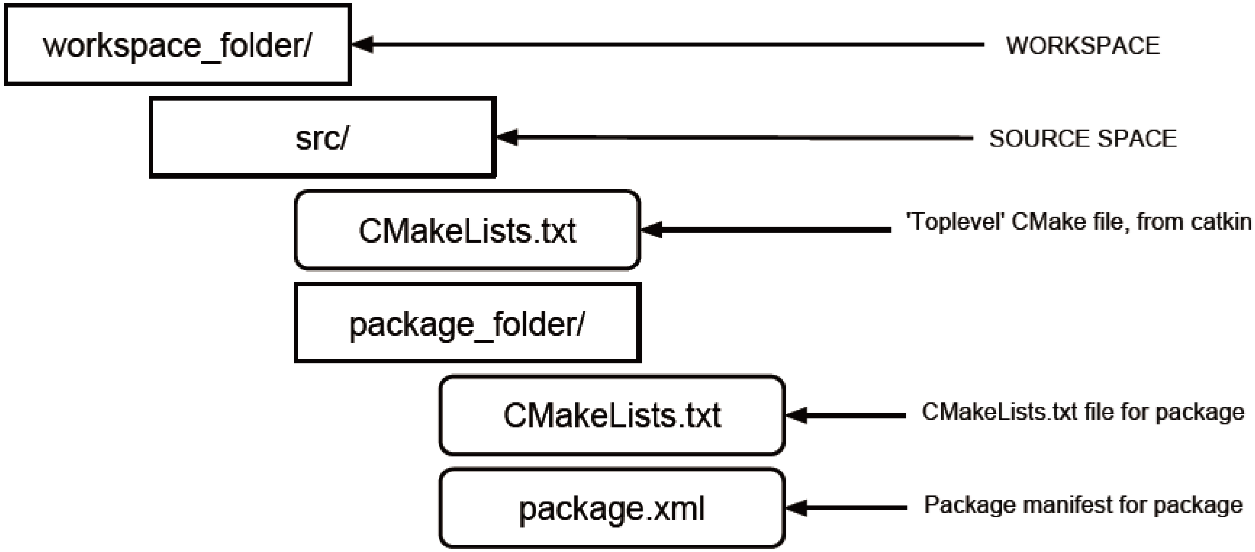
- Even though the workspace is empty (there are no packages in the 'src' folder, just a single CMakeLists.txt link) we can still "build" the workspace:

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

- catkin_make command builds the workspace and all the packages within it.

Creating a ROS Package

- **For a package to be considered a catkin package it must meet a few requirements:**
  - The package must contain a catkin compliant package.xml file
    - That package.xml file provides meta information about the package
  - The package must contain a CMakeLists.txt which uses catkin.
    - Catkin metapackages must have a boilerplate CMakeLists.txt file.
  - There can be no more than one package in each folder.
    - This means no nested packages nor multiple packages sharing the same directory



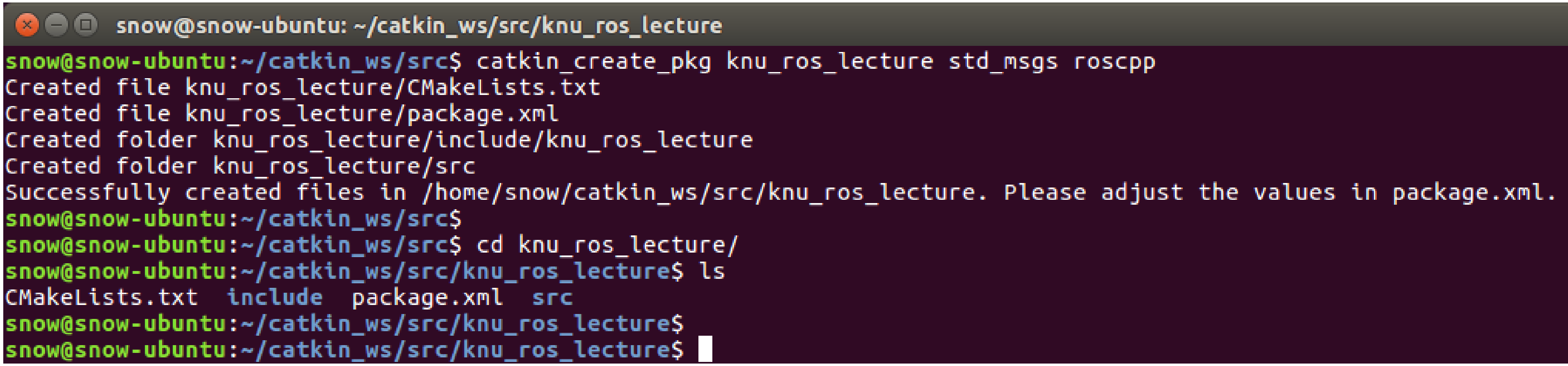
Creating a ROS Package

- **catkin_create_pkg**
  - Change to the source directory of the workspace

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

  - Creates a new package with the specified dependencies
  - Usage: `$ catkin_create_pkg <pkg_name> [depend1] [depend2] ...`
  - Try:

```
$ catkin_create_pkg knu_ros_lecture std_msgs roscpp
```



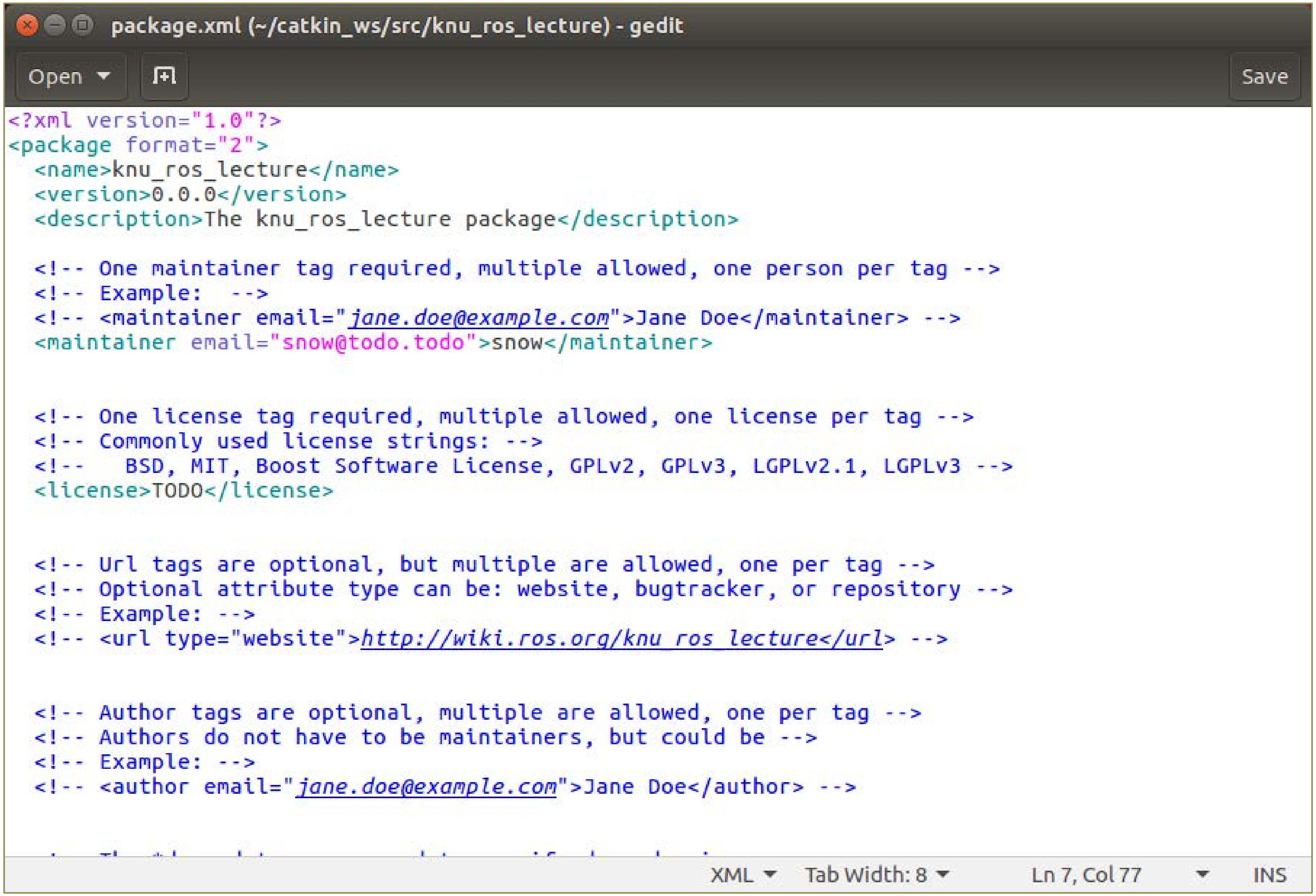
```
snow@snow-ubuntu:~/catkin_ws/src$ catkin_create_pkg knu_ros_lecture std_msgs roscpp
Created file knu_ros_lecture/CMakeLists.txt
Created file knu_ros_lecture/package.xml
Created folder knu_ros_lecture/include/knu_ros_lecture
Created folder knu_ros_lecture/src
Successfully created files in /home/snow/catkin_ws/src/knu_ros_lecture. Please adjust the values in package.xml.
snow@snow-ubuntu:~/catkin_ws/src$
snow@snow-ubuntu:~/catkin_ws/src$ cd knu_ros_lecture/
snow@snow-ubuntu:~/catkin_ws/src/knu_ros_lecture$ ls
CMakeLists.txt  include  package.xml  src
snow@snow-ubuntu:~/catkin_ws/src/knu_ros_lecture$
snow@snow-ubuntu:~/catkin_ws/src/knu_ros_lecture$ 
```

Creating a ROS Package

- std_msgs : a pre-defined structure for ROS data communication
- roscpp : a ROS client implementation in C++
  - Library documentation can be found at: http://docs.ros.org/api/roscpp/html/
  - ROS main header file is "ros/ros.h"
- package.xml file that defines properties about the package such as:
  - the package name
  - version numbers
  - authors
  - dependencies on other catkin packages

Creating a ROS Package

```
$ cd ~/catkin_ws/src/knu_ros_lecture
$ gedit package.xml
```



```
<?xml version="1.0"?>
<package format="2">
  <name>knu_ros_lecture</name>
  <version>0.0.0</version>
  <description>The knu_ros_lecture package</description>

  <!-- One maintainer tag required, multiple allowed, one person per tag -->
  <!-- Example:  -->
  <!-- <maintainer email="jane.doe@example.com">Jane Doe</maintainer> -->
  <maintainer email="snow@todo.todo">snow</maintainer>


  <!-- One license tag required, multiple allowed, one license per tag -->
  <!-- Commonly used license strings: -->
  <!--   BSD, MIT, Boost Software License, GPLv2, GPLv3, LGPLv2.1, LGPLv3 -->
  <license>TODO</license>


  <!-- Url tags are optional, but multiple are allowed, one per tag -->
  <!-- Optional attribute type can be: website, bugtracker, or repository -->
  <!-- Example: -->
  <!-- <url type="website">http://wiki.ros.org/knu_ros_lecture</url> -->


  <!-- Author tags are optional, multiple are allowed, one per tag -->
  <!-- Authors do not have to be maintainers, but could be -->
  <!-- Example: -->
  <!-- <author email="jane.doe@example.com">Jane Doe</author> -->
```

First Node Example : "Hello World! "

- Change to the source directory of our package knu_ros_lecture

```
$ cd ~/catkin_ws/src/knu_ros_lecture/src
```

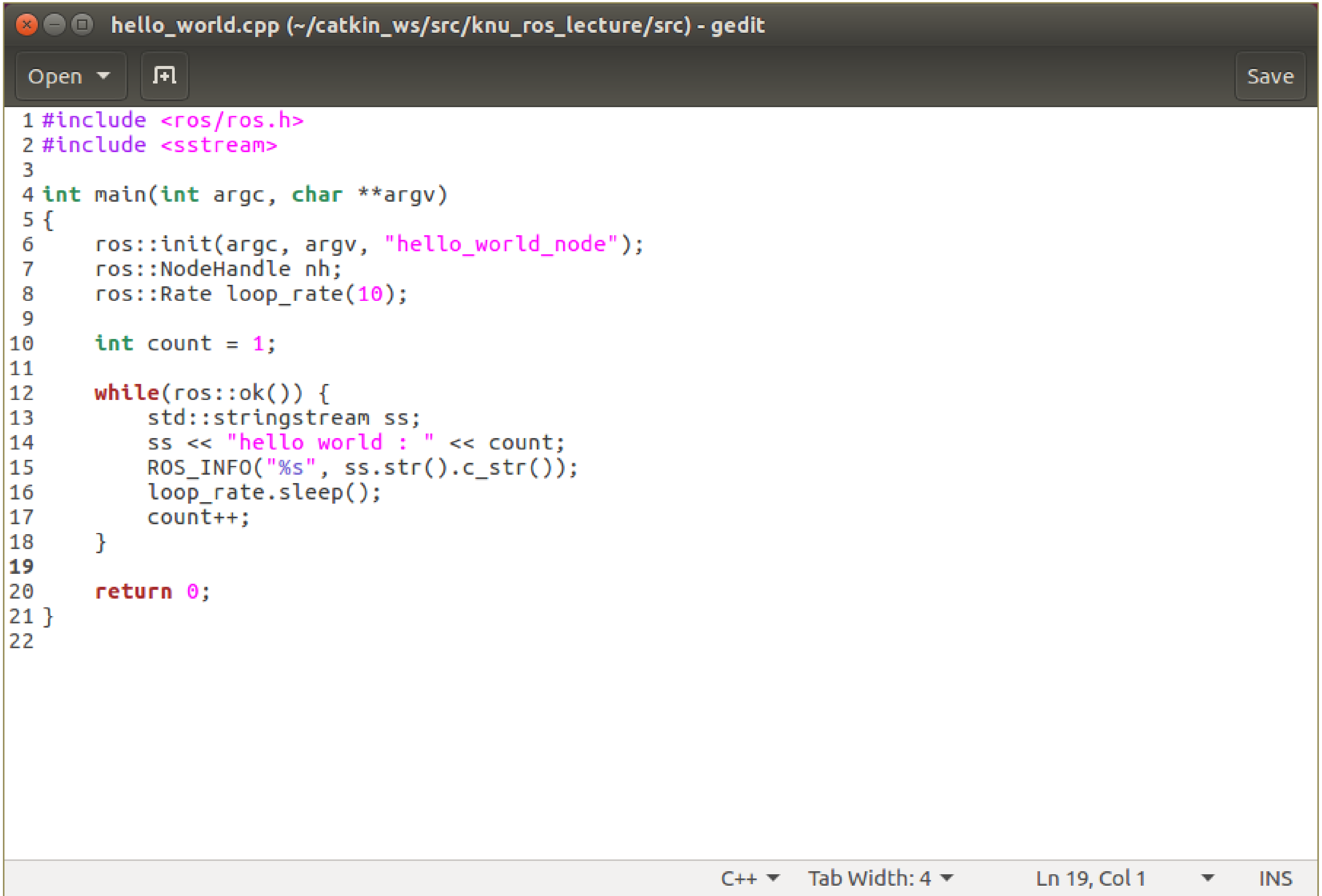
- Create & edit a source file (.cpp) with a text-editor

```
$ gedit hello_world.cpp
```

  - gedit
  - vi / vim
  - emacs
  - qtcreator
  - eclipse
  - etc.

First Node Example : "Hello World! "

- **Edit hello_world.cpp**



hello_world.cpp (~/catkin_ws/src/knu_ros_lecture/src) - gedit

```cpp
#include <ros/ros.h>
#include <sstream>

int main(int argc, char **argv)
{
    ros::init(argc, argv, "hello_world_node");
    ros::NodeHandle nh;
    ros::Rate loop_rate(10);

    int count = 1;

    while(ros::ok()) {
        std::stringstream ss;
        ss << "hello world : " << count;
        ROS_INFO("%s", ss.str().c_str());
        loop_rate.sleep();
        count++;
    }

    return 0;
}

```

C++ Tab Width: 4 Ln 19, Col 1 INS

First Node Example : "Hello World! "

- **hello_world.cpp**
  - ros::init() must be called before using any of the rest of the ROS system
  - Typical call in the main() function:

```
ros::init(argc, argv, "Node name");
```

  - “Node name” must be unique in a running system

First Node Example : "Hello World! "

- **hello_world.cpp**
  - ros::NodeHandle is main access point to communicate with the ROS system.
    - It provides public interface to topics, services, parameters, etc.
  - Create a handle to this process’ node (after the call to ros::init()) by declaring:

```
ros::NodeHandle nh;
```

    - The first NodeHandle constructed will fully initialize the current node
    - The last NodeHandle destructed will close down the node

First Node Example : "Hello World! "

hello_world.cpp

- ros::Rate is a class to help run loops at a desired frequency.
- Specify in the c’tor the destired rate to run in Hz

```
ros::Rate loop_rate(10);
```

- ros::Rate::sleep() method
  - Sleeps for any leftover time in a cycle.
  - Calculated from the last time sleep, reset, or the constructor was called

First Node Example : "Hello World! "

- **hello_world.cpp**
  - Call ros::ok() to check if the node should continue running
  - ros::ok() will return false if:
    - a SIGINT is received (Ctrl-C)
    - We have been kicked off the network by another node with the same name
    - ros::shutdown() has been called by another part of the application.
    - all ros::NodeHandles have been destroyed

First Node Example : "Hello World! "

- **hello_world.cpp**
  - ROS_INFO prints an informative message
    - ROS_INFO("My INFO message.");
  - This function allows parameters as in printf:
    - ROS_INFO("My INFO message with argument: %f", val);

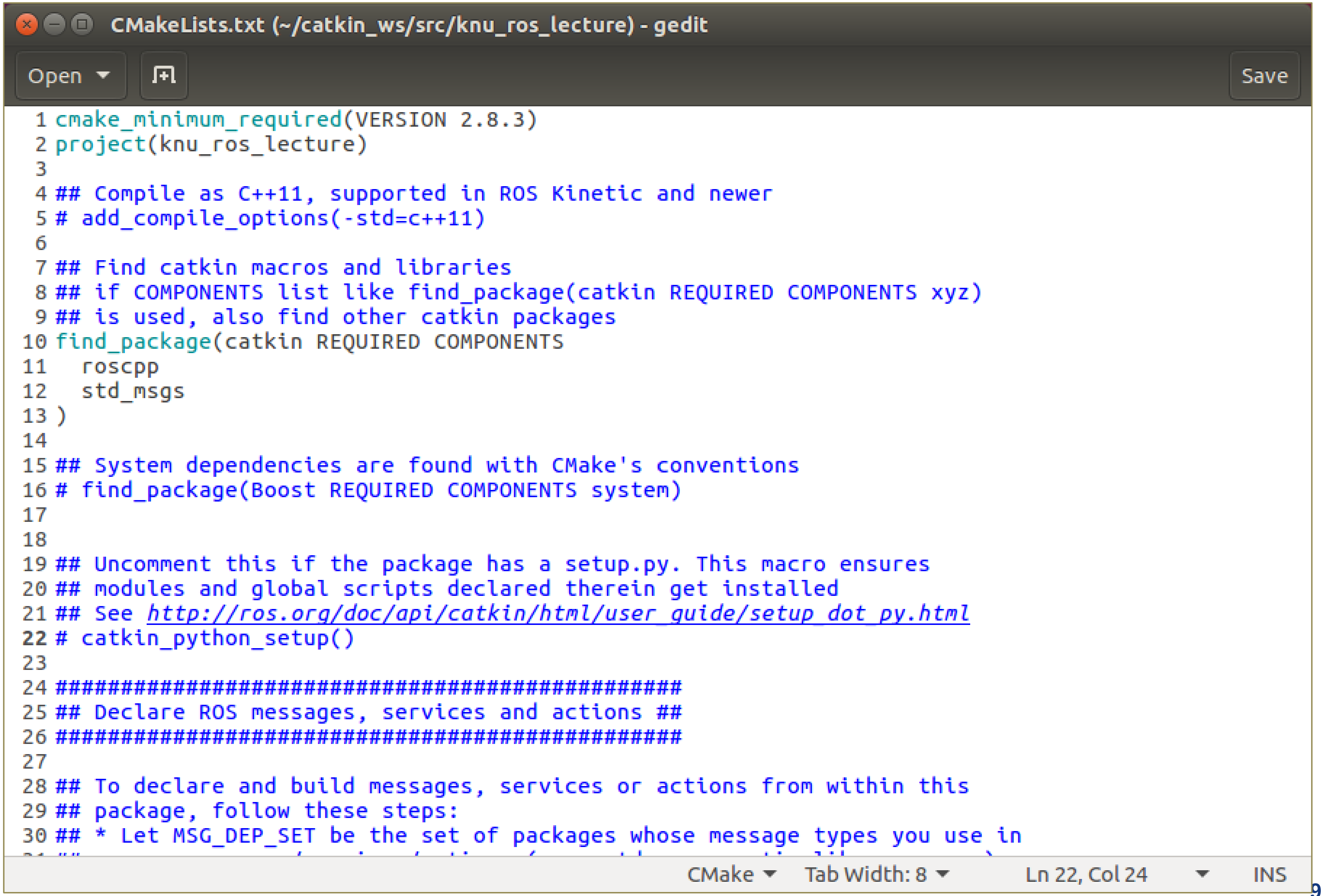
First Node Example : "Hello World! "

- **Edit CMakeLists.txt**

```
$ gedit ~/catkin_ws/src/knu_ros_lecture/CMakeLists.txt
```

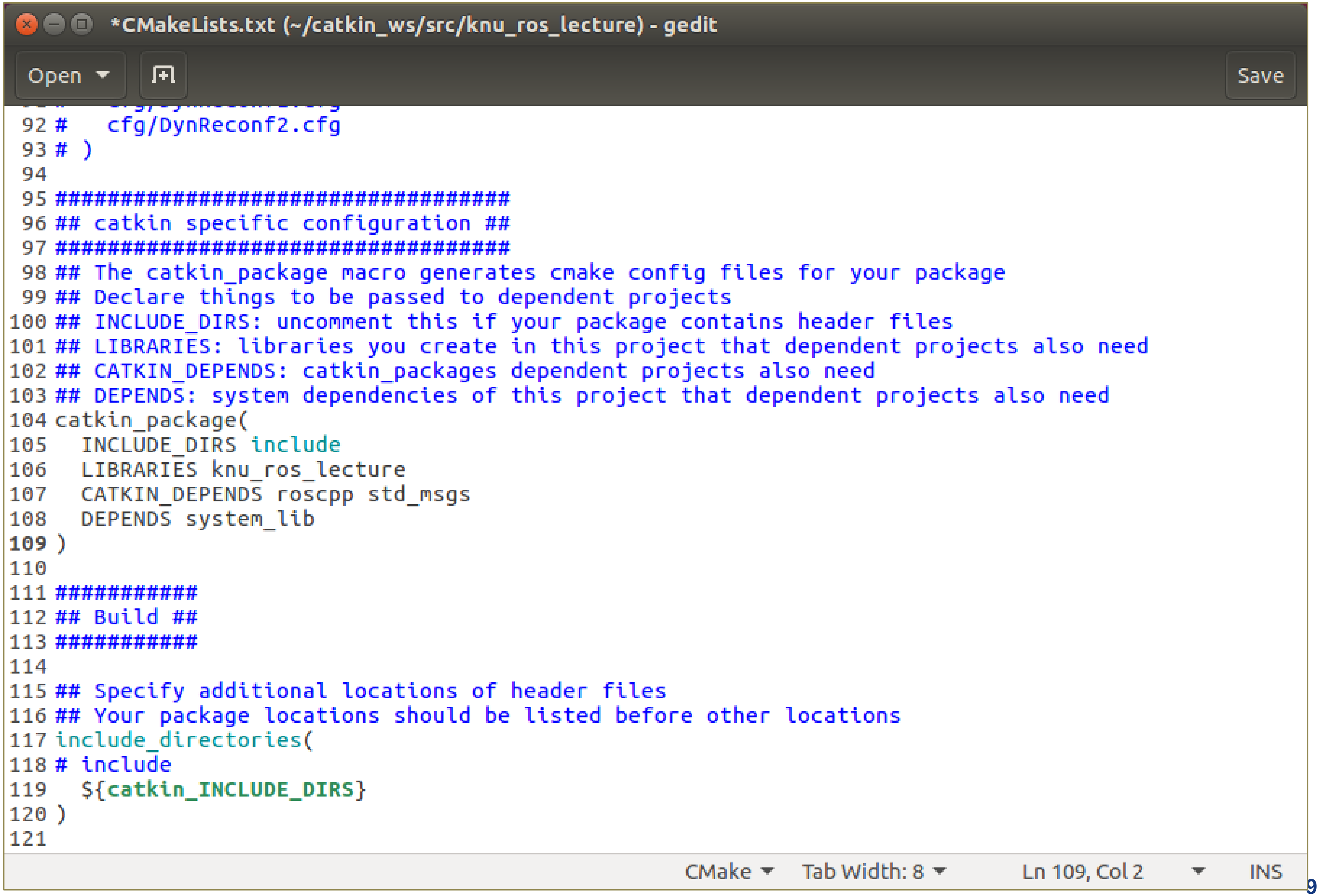
```
cmake_minimum_required(VERSION 2.8.3)
project(knu_ros_lecture)

find_package(catkin REQUIRED COMPONENTS
  roscpp
  std_msgs
)

catkin_package(
  INCLUDE_DIRS include
  LIBRARIES knu_ros_lecture
  CATKIN_DEPENDS roscpp std_msgs
  DEPENDS system_lib
)

include_directories(${catkin_INCLUDE_DIRS})

add_executable(hello_world_node src/hello_world.cpp)
add_dependencies(hello_world_node knu_ros_lecture_generate_messages_cpp)
target_link_libraries(hello_world_node ${catkin_LIBRARIES})
```

CMakeLists.txt



```
cmake_minimum_required(VERSION 2.8.3)
project(knu_ros_lecture)

## Compile as C++11, supported in ROS Kinetic and newer
# add_compile_options(-std=c++11)

## Find catkin macros and libraries
## if COMPONENTS list like find_package(catkin REQUIRED COMPONENTS xyz)
## is used, also find other catkin packages
find_package(catkin REQUIRED COMPONENTS
  roscpp
  std_msgs
)

## System dependencies are found with CMake's conventions
# find_package(Boost REQUIRED COMPONENTS system)


## Uncomment this if the package has a setup.py. This macro ensures
## modules and global scripts declared therein get installed
## See http://ros.org/doc/api/catkin/html/user_guide/setup_dot_py.html
# catkin_python_setup()

################################################
## Declare ROS messages, services and actions ##
################################################

## To declare and build messages, services or actions from within this
## package, follow these steps:
## * Let MSG_DEP_SET be the set of packages whose message types you use in
```

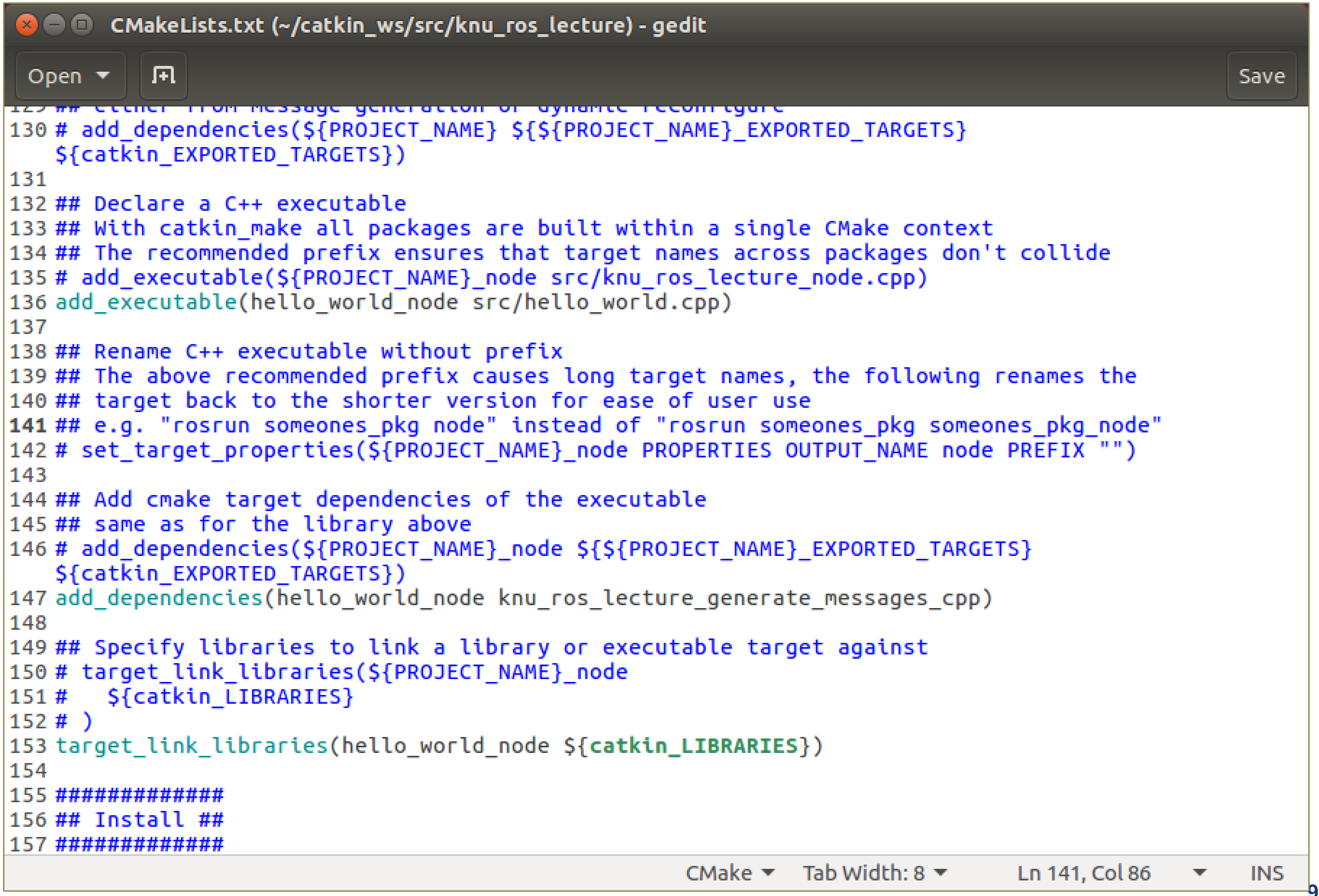
CMakeLists.txt



```
#   cfg/DynReconf2.cfg
# )

###################################
## catkin specific configuration ##
###################################
## The catkin_package macro generates cmake config files for your package
## Declare things to be passed to dependent projects
## INCLUDE_DIRS: uncomment this if your package contains header files
## LIBRARIES: libraries you create in this project that dependent projects also need
## CATKIN_DEPENDS: catkin_packages dependent projects also need
## DEPENDS: system dependencies of this project that dependent projects also need
catkin_package(
  INCLUDE_DIRS include
  LIBRARIES knu_ros_lecture
  CATKIN_DEPENDS roscpp std_msgs
  DEPENDS system_lib
)

###########
## Build ##
###########

## Specify additional locations of header files
## Your package locations should be listed before other locations
include_directories(
# include
  ${catkin_INCLUDE_DIRS}
)

```

CMake ▾ Tab Width: 8 ▾ Ln 109, Col 2 ▾ INS

CMakeLists.txt



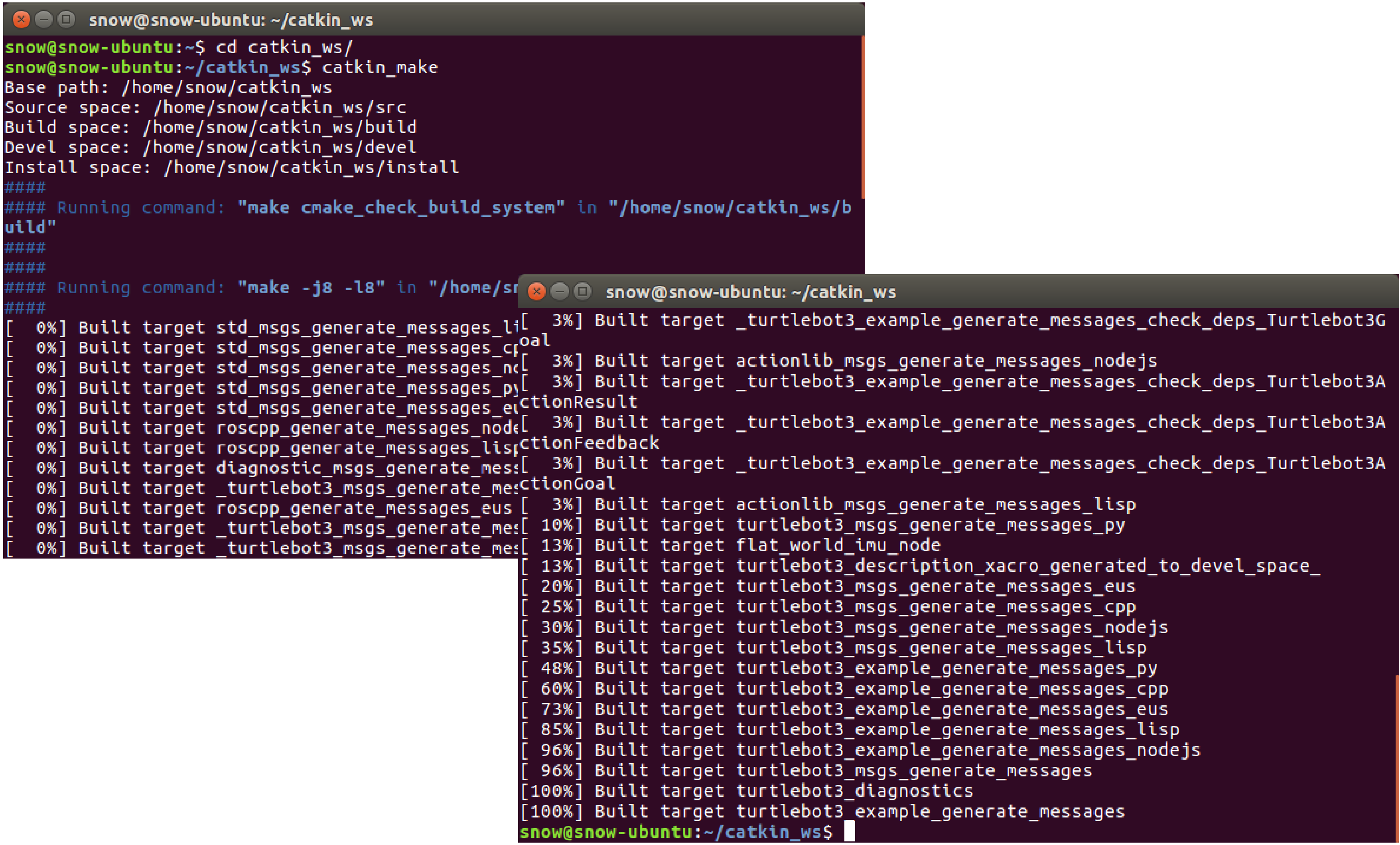
```
## either from message generation or dynamic reconfigure
# add_dependencies(${PROJECT_NAME} ${${PROJECT_NAME}_EXPORTED_TARGETS} ${catkin_EXPORTED_TARGETS})

## Declare a C++ executable
## With catkin_make all packages are built within a single CMake context
## The recommended prefix ensures that target names across packages don't collide
# add_executable(${PROJECT_NAME}_node src/knu_ros_lecture_node.cpp)
add_executable(hello_world_node src/hello_world.cpp)

## Rename C++ executable without prefix
## The above recommended prefix causes long target names, the following renames the
## target back to the shorter version for ease of user use
## e.g. "rosrun someones_pkg node" instead of "rosrun someones_pkg someones_pkg_node"
# set_target_properties(${PROJECT_NAME}_node PROPERTIES OUTPUT_NAME node PREFIX "")

## Add cmake target dependencies of the executable
## same as for the library above
# add_dependencies(${PROJECT_NAME}_node ${${PROJECT_NAME}_EXPORTED_TARGETS} ${catkin_EXPORTED_TARGETS})
add_dependencies(hello_world_node knu_ros_lecture_generate_messages_cpp)

## Specify libraries to link a library or executable target against
# target_link_libraries(${PROJECT_NAME}_node
#   ${catkin_LIBRARIES}
# )
target_link_libraries(hello_world_node ${catkin_LIBRARIES})

#############
## Install ##
#############
```

CMake ▾ Tab Width: 8 ▾ Ln 141, Col 86 ▾ INS

Building Your Nodes

- **To build the package in the terminal call catkin_make**

```
$ cd ~/catkin_ws && catkin_make
```



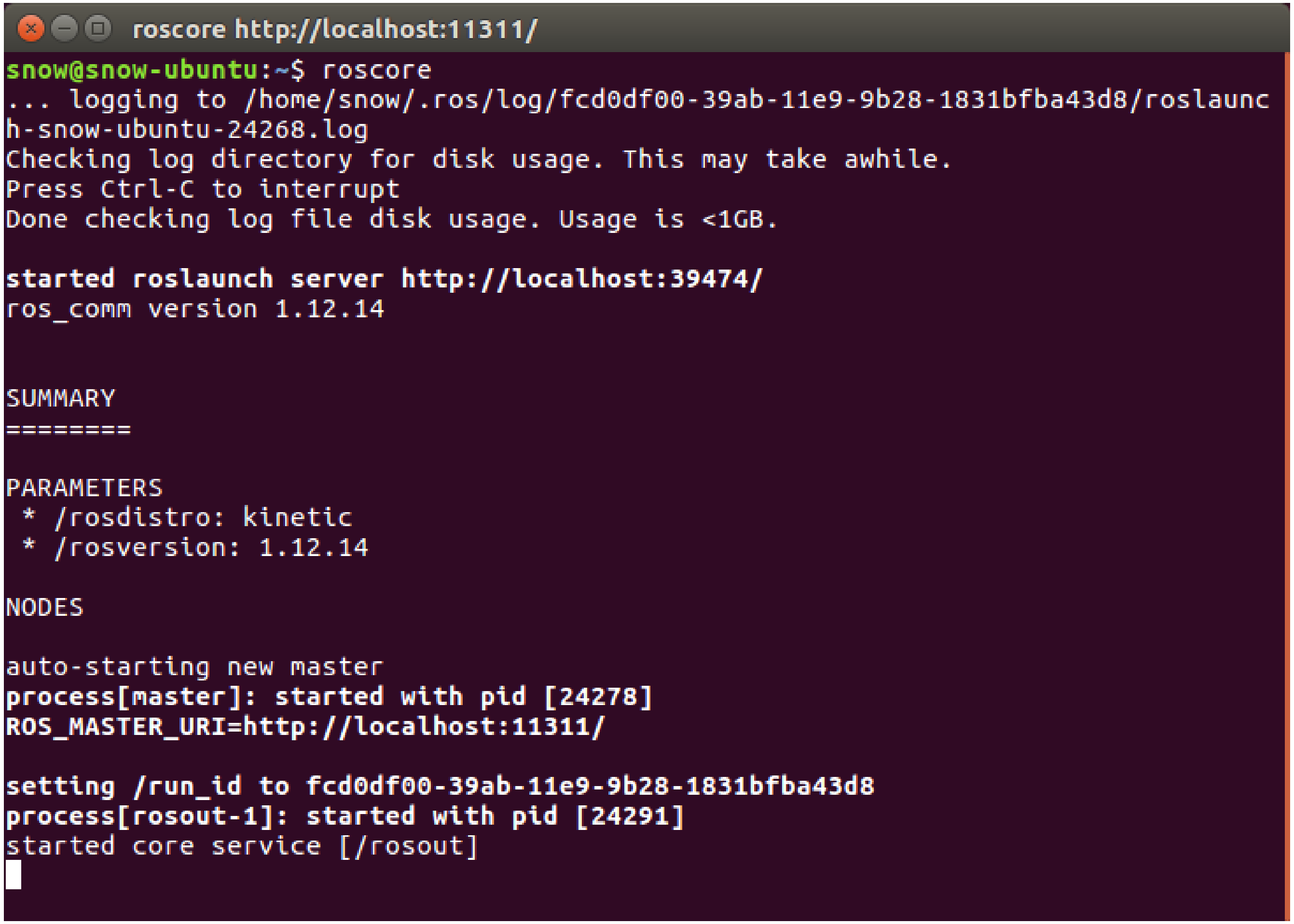
```
snow@snow-ubuntu:~$ cd catkin_ws/
snow@snow-ubuntu:~/catkin_ws$ catkin_make
Base path: /home/snow/catkin_ws
Source space: /home/snow/catkin_ws/src
Build space: /home/snow/catkin_ws/build
Devel space: /home/snow/catkin_ws/devel
Install space: /home/snow/catkin_ws/install
####
#### Running command: "make cmake_check_build_system" in "/home/snow/catkin_ws/b
uild"
####
####
#### Running command: "make -j8 -l8" in "/home/sr
####
[  0%] Built target std_msgs_generate_messages_li
[  0%] Built target std_msgs_generate_messages_cp
[  0%] Built target std_msgs_generate_messages_nc
[  0%] Built target std_msgs_generate_messages_py
[  0%] Built target std_msgs_generate_messages_eu
[  0%] Built target roscpp_generate_messages_node
[  0%] Built target roscpp_generate_messages_lisp
[  0%] Built target diagnostic_msgs_generate_mess
[  0%] Built target _turtlebot3_msgs_generate_mes
[  0%] Built target roscpp_generate_messages_eus
[  0%] Built target _turtlebot3_msgs_generate_mes
[  0%] Built target _turtlebot3_msgs_generate_mes
```

```
[  3%] Built target _turtlebot3_example_generate_messages_check_deps_Turtlebot3G
oal
[  3%] Built target actionlib_msgs_generate_messages_nodejs
[  3%] Built target _turtlebot3_example_generate_messages_check_deps_Turtlebot3A
ctionResult
[  3%] Built target _turtlebot3_example_generate_messages_check_deps_Turtlebot3A
ctionFeedback
[  3%] Built target _turtlebot3_example_generate_messages_check_deps_Turtlebot3A
ctionGoal
[  3%] Built target actionlib_msgs_generate_messages_lisp
[ 10%] Built target turtlebot3_msgs_generate_messages_py
[ 13%] Built target flat_world_imu_node
[ 13%] Built target turtlebot3_description_xacro_generated_to_devel_space_
[ 20%] Built target turtlebot3_msgs_generate_messages_eus
[ 25%] Built target turtlebot3_msgs_generate_messages_cpp
[ 30%] Built target turtlebot3_msgs_generate_messages_nodejs
[ 35%] Built target turtlebot3_msgs_generate_messages_lisp
[ 48%] Built target turtlebot3_example_generate_messages_py
[ 60%] Built target turtlebot3_example_generate_messages_cpp
[ 73%] Built target turtlebot3_example_generate_messages_eus
[ 85%] Built target turtlebot3_example_generate_messages_lisp
[ 96%] Built target turtlebot3_example_generate_messages_nodejs
[ 96%] Built target turtlebot3_msgs_generate_messages
[100%] Built target turtlebot3_diagnostics
[100%] Built target turtlebot3_example_generate_messages
snow@snow-ubuntu:~/catkin_ws$
```

Running Your Nodes in Terminal

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

```
$ roscore
```

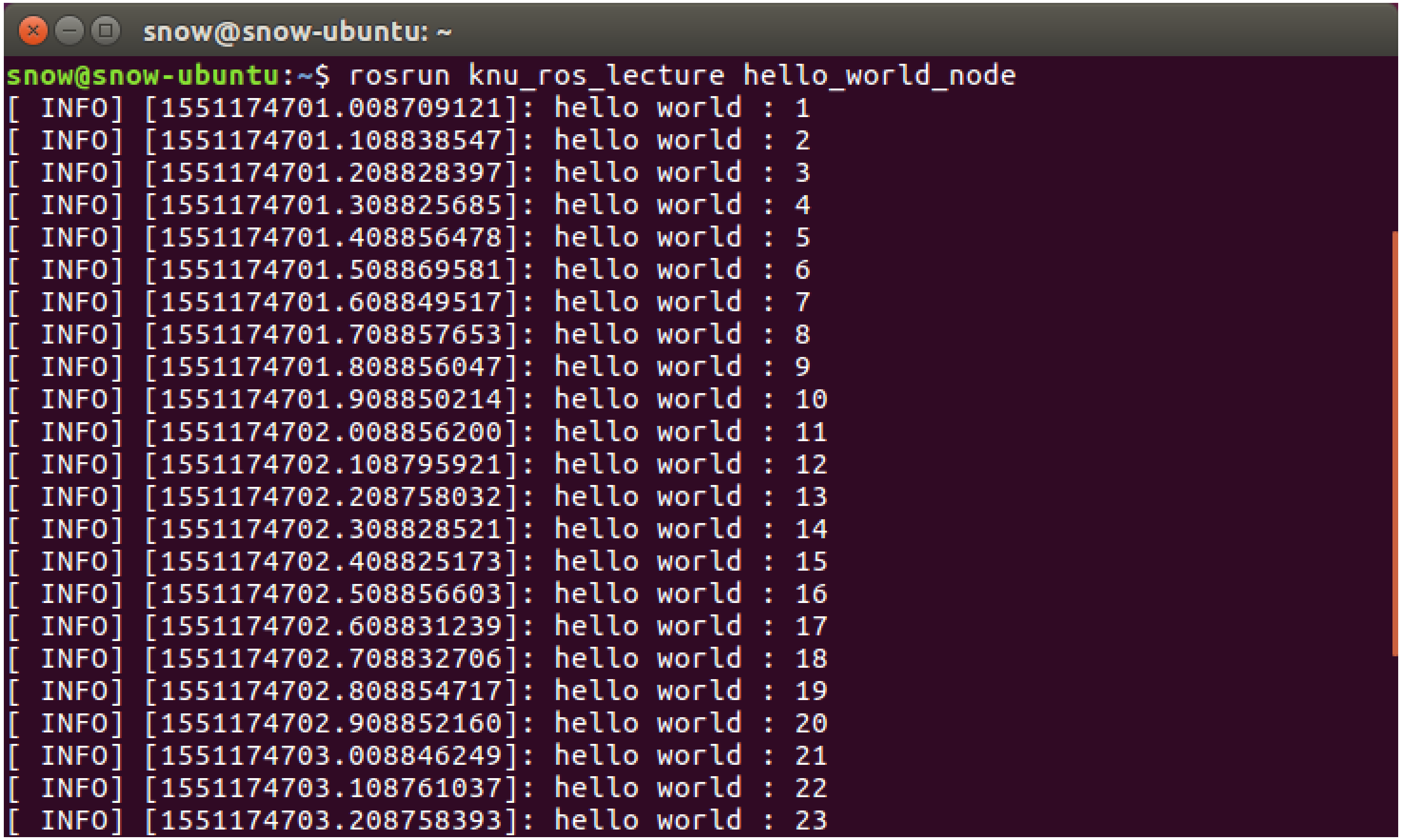


```
snow@snow-ubuntu:~$ roscore
... logging to /home/snow/.ros/log/fcd0df00-39ab-11e9-9b28-1831bfba43d8/roslaunc
h-snow-ubuntu-24268.log
Checking log directory for disk usage. This may take awhile.
Press Ctrl-C to interrupt
Done checking log file disk usage. Usage is <1GB.

started roslaunch server http://localhost:39474/
ros_comm version 1.12.14


SUMMARY
========

PARAMETERS
 * /rosdistro: kinetic
 * /rosversion: 1.12.14

NODES

auto-starting new master
process[master]: started with pid [24278]
ROS_MASTER_URI=http://localhost:11311/

setting /run_id to fcd0df00-39ab-11e9-9b28-1831bfba43d8
process[rosout-1]: started with pid [24291]
started core service [/rosout]
```

Running Your Nodes in Terminal

- **rosrun allows you to use the package name to directly run a node within a package. (new Terminal)**

```
$ rosrun knu_ros_lecture hello_world_node
```



```
snow@snow-ubuntu:~$ rosrun knu_ros_lecture hello_world_node
[ INFO] [1551174701.008709121]: hello world : 1
[ INFO] [1551174701.108838547]: hello world : 2
[ INFO] [1551174701.208828397]: hello world : 3
[ INFO] [1551174701.308825685]: hello world : 4
[ INFO] [1551174701.408856478]: hello world : 5
[ INFO] [1551174701.508869581]: hello world : 6
[ INFO] [1551174701.608849517]: hello world : 7
[ INFO] [1551174701.708857653]: hello world : 8
[ INFO] [1551174701.808856047]: hello world : 9
[ INFO] [1551174701.908850214]: hello world : 10
[ INFO] [1551174702.008856200]: hello world : 11
[ INFO] [1551174702.108795921]: hello world : 12
[ INFO] [1551174702.208758032]: hello world : 13
[ INFO] [1551174702.308828521]: hello world : 14
[ INFO] [1551174702.408825173]: hello world : 15
[ INFO] [1551174702.508856603]: hello world : 16
[ INFO] [1551174702.608831239]: hello world : 17
[ INFO] [1551174702.708832706]: hello world : 18
[ INFO] [1551174702.808854717]: hello world : 19
[ INFO] [1551174702.908852160]: hello world : 20
[ INFO] [1551174703.008846249]: hello world : 21
[ INFO] [1551174703.108761037]: hello world : 22
[ INFO] [1551174703.208758393]: hello world : 23
```

Any Questions…
Just Ask!