Chapter 5

ROS Commands

5.1. ROS Command List

ROS Wiki

The ROS commands are explained in detail on the Wiki page 'http://wiki.ros.org/ROS/Command LineTools'. In addition, the GitHub repository 'https://github.com/ros/cheatsheet/releases' summarizes important commands described in this chapter. It will be a helpful reference to utilize along with the descriptions in this chapter.

When using ROS, we enter commands in a Shell environment to perform tasks such as using file systems, editing, building, debugging source codes, package management, etc. In order to use ROS properly, we will need to familiarize ourselves with not only the basic Linux commands but also with the ROS specific commands.

In order to become proficient with the various commands used for ROS, we will give a brief description of the functions of each command, and introduce them one by one with examples. When introducing a command, each one is ranked with three stars based on their frequency of use and importance. You may need some time to get used to the commands, but the more you use them, you will soon find yourself using each kind of ROS functions quickly and easily using these commands.

ROS Shell Commands

Command	Importance	Command Explanation	Description
roscd	★★★	ros+cd(changes directory)	Move to the directory of the designated ROS package
rosls	★☆☆	ros+ls(lists files)	Check file list of ROS package
rosed	★☆☆	ros+ed(editor)	Edit file of ROS package
roscp	★☆☆	ros+cp(copies files)	Copy file of ROS package
rospd	☆☆☆	ros+pushd	Add directory to the ROS directory index
rosd	☆☆☆	ros+directory	Check the ROS directory index

ROS Execution Commands

Command	Importance	Command Explanation	Description
roscore	★★★	ros+core	master(ROS name service) + rosout(record log) + parameter server(manage parameter)

Command	Importance	Command Explanation	Description
rosrun	★★★	ros+run	Run node
roslaunch	★★★	ros+launch	Launch multiple nodes and configure options
rosclean	★★☆	ros+clean	Examine or delete ROS log file

ROS Information Commands

Command	Importance	Command Explanation	Description
rostopic	★★★	ros+topic	Check ROS topic information
rosservice	★★★	ros+service	Check ROS service information
rosnode	★★★	ros+node	Check ROS node information
rosparam	★★★	ros+param(parameter)	Check and edit ROS parameter information
rosbag	★★★	ros+bag	Record and play ROS message
rosmsg	★★☆	ros+msg	Check ROS message information
rossrv	★★☆	ros+srv	Check ROS service information
rosversion	★☆☆	ros+version	Check ROS package and release version information
roswtf	☆☆☆	ros+wtf	Examine ROS system

ROS Catkin Commands

Command	Importance	Description
catkin_create_pkg	★★★	Automatic creation of package
catkin_make	★★★	Build based on catkin build system
catkin_eclipse	★★☆	Modify package created by catkin build system so that it can be used in Eclipse
catkin_prepare_release	★★☆	Cleanup log and tag version during release
catkin_generate_changelog	★★☆	Create or update 'CHANGELOG.rst' file during release
catkin_init_workspace	★★☆	Initialize workspace of the catkin build system
catkin_find	★☆☆	Search catkin

ROS Package Commands

Command	Importance	Command Explanation	Description
ropack	★★★	ros+pack(package)	View information regarding a specific ROS package
rosinstall	★★☆	ros+install	Install additional ROS packages
rosdep	★★☆	ros+dep(dependencies)	Install dependency package of the ROS corresponding package
roslocate	☆☆☆	ros+locate	Show information of ROS package
roscreate-pkg	☆☆☆	ros+create-pkg(package)	Automatic creation of ROS package (used in previous rosbuild system)
rosmake	☆☆☆	ros+make	Build ROS package (used in previous rosbuild system)

5.2. ROS Shell Commands

ROS shell commands are also called rosbash[1]. These commands allow us to use the bash shell commands commonly used in Linux for the ROS development environment as well. Generally 'ros' prefixes are used in conjunction with suffixes such as 'cd, pd, d, ls, ed, cp, run'. The relevant commands are as follows.

Command	Importance	Command Explanation	Description
roscd	★★★	ros+cd(changes directory)	Move to the directory of the designated ROS package
rosls	★☆☆	ros+ls(lists files)	Check file list of ROS package
rosed	★☆☆	ros+ed(editor)	Edit file of ROS package
roscp	★☆☆	ros+cp(copies files)	Copy file of ROS package
rospd	☆☆☆	ros+pushd	Add directory to the ROS directory index
rosd	☆☆☆	ros+directory	Check the ROS directory index

From these, we will look into the commands 'roscd', 'rosls', and 'rosed' which are used frequently.

1 http://wiki.ros.org/rosbash

Environment for using ROS shell commands

In order to use ROS shell commands, 'rosbash' must be installed using the following command, and can only be used in a terminal window that has configured 'source /opt/ros/<ros distribution>/setup.bash'. This does not have to be installed separately, and if you have finished building the ROS development environment from Chapter 3 then you will be able to use it.

```
$ sudo apt-get install ros-<ros distribution>-rosbash
```

5.2.1. roscd: ROS Change Directory

```
roscd [PACKAGE_NAME]
```

This is a command to move to the directory where the package is saved. The basic instruction is to type the 'roscd' command followed by the package name as a parameter. In the following example, the turtlesim package is in the folder where ROS is installed so we get the following result, but if you put the name of a package that you created (for example, my_first_ros_pkg created in Chapter 4) as a parameter, then it moves to the folder of the package that you have designated. This is a command that is frequently used in the command-based ROS.

```
$ roscd turtlesim
/opt/ros/kinetic/share/turtlesim $
$ roscd my_first_ros_pkg
~/catkin_ws/src/my_first_ros_pkg $
```

The ros-kinetic-turtlesim package must be installed first in order to get the identical result as shown above. If it is not installed yet, you can install it with the following command.

```
$ sudo apt-get install ros-kinetic-turtlesim
```

If the package is already installed, you will see the message that says the package is already installed as shown below.

```
$ sudo apt-get install ros-kinetic-turtlesim
[sudo] password for USER:
Reading package lists... Done
Building dependency tree
Reading state information... Done
ros-kinetic-turtlesim is already the newest version (0.7.1-0xenial-20170613-170649-0800).
0 upgraded, 0 newly installed, 0 to remove and 29 not upgraded.
```

5.2.2. rosls: ROS File List

```
rosls [PACKAGE_NAME]
```

This is a command to check the file list of the specific ROS package. We can use the 'roscd' command to move to the corresponding package folder and then use the 'ls' command to perform the same function, but this command is used occasionally when we need to check without moving to the package directory.

```
$ rosls turtlesim
cmake images msg srv package.xml
```

5.2.3. rosed: ROS Edit Command

```
rosed [PACKAGE_NAME] [FILE_NAME]
```

This is a command used to edit a specific file in the package. If you run this command, it opens the corresponding file with the editor that the user has set up. This is often used when you want to quickly make a simple modification. The user can assign which editor will be used by editing the command export EDITOR = 'emacs -nw' in your '~/.bashrc' file. As previously mentioned, this command is used for simple tasks that need to modify directly in the command window, and it is not recommended for complex tasks. It is not a command that is used often.

```
$ rosed turtlesim package.xml
```

5.3. ROS Execution Commands

The ROS execution commands control the execution of ROS nodes. Above all, the most essential is roscore which is used as the name server for nodes. And for the execution commands there are 'rosrun' and 'roslaunch'. We use 'rosrun' to run one node and 'roslaunch' to run multiple nodes or to additionally configure various options. And 'rosclean' is a command that deletes the logs recorded when node was running.

Command	Importance	Command Explanation	Description
roscore	★★★	ros+core	master(ROS name service) + rosout(record log) + parameter server(manage parameter)
rosrun	★★★	ros+run	Run node

Command	Importance	Command Explanation	Description
roslaunch	★★★	ros+launch	Launch multiple nodes and configure options
rosclean	★★☆	ros+clean	Examine or delete ROS log file

5.3.1. roscore: Run roscore

```
roscore [OPTION]
```

Roscore is the master that manages the connection information for communication among nodes, and is an essential element that must be the first to be launched to use ROS. The ROS master is launched by the 'roscore' command, and runs as an XMLRPC server. The master registers node information such as names, topics and service names, message types, URI addresses and ports, and when there is a request this information is passed to other nodes. Upon the launch of 'roscore', 'rosout' is executed as well, which is used to record ROS standard output logs such as DEBUG, INFO, WARN, ERROR, FATAL, etc. Also the parameter server which manages the parameters is executed.

When roscore is running, the URI configured in the ROS_MASTER_URI is set as the master URI to run the master. ROS_MASTER_URI can be configured by the user in '~/.bashrc' file as mentioned in the ROS Configuration section in Chapter 3.

```
$ roscore
... logging to /home/pyo/.ros/log/c2d0b528-6536-11e7-935b-08d40c80c500/roslaunch-pyo-20002.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:43517/
ros_comm version 1.12.7

SUMMARY
========
PARAMETERS
 * /rosdistro: kinetic
 * /rosversion: 1.12.7

NODES
auto-starting new master
```

```
process[master]: started with pid [20013]
ROS_MASTER_URI=http://localhost:11311/

setting /run_id to c2d0b528-6536-11e7-935b-08d40c80c500
process[rosout-1]: started with pid [20027]
started core service [/rosout]
```

In the terminal screen, we can see that the logs are saved in the directory '/home/xxx/.ros/log/'. The displayed message also notifies that we can close roscore with [Ctrl+c], the information of roslaunch server and ROS_MASTER_URI, and that the parameter server of '/rosdistro' and '/rosversion' and the /rosout node have all been running.

Log Save Path

In the above example, it shows that the path where the logs are saved is '/home/xxx/.ros/log/', but in reality it is saved in the location where the ROS_HOME environment variable is configured. If the ROS_HOME environment variable has not been configured, then the default value is '~/.ros/log/'.

5.3.2. rosrun: Run ROS Node

```
rosrun [PACKAGE_NAME] [NODE_NAME]
```

Rosrun is a command that runs only one node in the specified package. The following example is a command that runs the 'turtlesim_node' node in the turtlesim package. For your information, the turtle icon that appears on the screen is selected randomly.

```
$ rosrun turtlesim turtlesim_node
[ INFO] [1512634911.275228307]: Starting turtlesim with node name /turtlesim
[ INFO] [1512634911.281614642]: Spawning turtle [turtle1] at x=[5.544445], y=[5.544445],
theta=[0.000000]
```

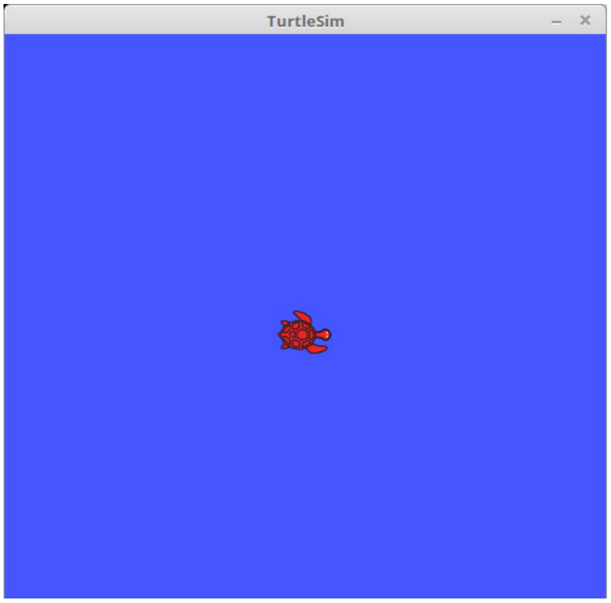

FIGURE 5-1 Screen after the turtlesim_node node is executed

5.3.3. roslaunch: Launch Multiple Nodes

```
roslaunch [PACKAGE_NAME] [launch_FILE_NAME]
```

Roslaunch is a command that executes more than one node in the specified package or sets execution options. As shown in the following example, simply launching the 'openni_launch' package will run more than 20 nodes and more than 10 parameter servers, such as 'camera_nodelet_manager', 'depth_metric', 'depth_metric_rect', 'depth_points', etc. As we can see, using the launch file is quite useful for running multiple nodes at the same time, and is a frequently used execution method in ROS. More information about creating the '*.launch' file that is used in this example will be provided in Section 7.6 Using roslaunch.

```
$ roslaunch openni_launch openni.launch
~ omitted ~
```

Note that in order to run this example and get the same result, the relevant package 'ros-kinetic-openni-launch' must be installed. If it is not installed yet, you can install it with the following command.

```
$ sudo apt-get install ros-kinetic-openni-launch
```

5.3.4. rosclean: Examine and Delete ROS Logs

```
rosclean [OPTION]
```

This is a command to check or delete the ROS log file. As ‘roscore’ is launched, the history of all nodes is recorded in the log file and the data accumulates over time, so it needs to be periodically deleted using the ‘rosclean’ command.

The following is an example for examining the log usage.

```
$ rosclean check
320K ROS node logs    → This means the total usage for the ROS node is 320KB
```

When running ‘roscore’, if the following WARNING message appears, it means that the log file exceeds 1GB. If the system is running out of space for the log, clean up the space with ‘rosclean’ command.

```
WARNING: disk usage in log directory [/xxx/.ros/log] is over 1GB.
```

The following is an example of deleting logs in the ROS log repository (it is ‘/home/rt/.ros/log’ in this example). If you wish to delete, press the ‘y’ key to proceed.

```
$ rosclean purge
Purging ROS node logs.
PLEASE BE CAREFUL TO VERIFY THE COMMAND BELOW!
Okay to perform:

rm -rf /home/pyo/.ros/log
(y/n)?
```

5.4. ROS Information Commands

ROS information commands are used to check information regarding topics, services, nodes, parameters, etc. In particular, ‘rostopic’, ‘rosservice’, ‘rosnode’, and ‘rosparam’ are used frequently, and ‘rosbag’ can record and play data, which is one of the major features of ROS, so be sure to fully understand it.

Command	Importance	Command Explanation	Description
rostopic	★★★	ros+topic	Check ROS topic information
rosservice	★★★	ros+service	Check ROS service information
rosnode	★★★	ros+node	Check ROS node information
rosparam	★★★	ros+param(parameter)	Check and edit ROS parameter information
rosbag	★★★	ros+bag	Record and play ROS message
rosmsg	★★☆	ros+msg	Check ROS message information
rossrv	★★☆	ros+srv	Check ROS service information
rosversion	★☆☆	ros+version	Check ROS package and release version information
roswtf	☆☆☆	ros+wtf	Examine ROS system

5.4.1. Run Node

We will use the following commands to practice the turtlesim provided by ROS in order to learn about nodes, topics and services. Prior to testing with the ROS information commands, we will need to make the following preparations.

Run roscore

Close all the terminals to ensure any conflicts with other processes. Then open a new terminal and run the following command.

```
$ roscore
```

In order to run the 'turtlesim_node' in the 'turtlesim' package, open a new terminal and run the following command. This will run 'turtlesim_node' in the 'turtlesim' package. A blue screen with a random turtle image will appear.

```
$ rosrun turtlesim turtlesim_node
```

Run the turtle_teleop_key node in the turtlesim package

Open a new terminal and run the following command. This will run the 'turtle_teleop_key' node in the 'turtlesim' package. Once it is run, we can use arrow keys on this terminal window to control the turtle. Pressing arrow keys will move the turtle on the screen. Although this is a simple simulation, it is still sending a message with the translational speed (m/s) and rotational speed (rad/s) necessary for moving the turtle, and we will be able to control an actual robot later with the same message remotely. For more detailed information and instructions regarding messages, refer to Section 4.2 Message Communication and Chapter 7 Basic ROS Programming.

```
$ rosrun turtlesim turtle_teleop_key
```

5.4.2. rosnode: ROS Node

Understanding nodes is necessary, so please refer to the Section 4.1 ROS Terminology.

Command	Description
rosnode list	Check the list of active nodes
rosnode ping [NODE_NAME]	Test connection with a specific node
rosnode info [NODE_NAME]	Check information of a specific node
rosnode machine [PC_NAME OR IP]	Check the list of nodes running on the corresponding PC
rosnode kill [NODE_NAME]	Stop running a specific node
rosnode cleanup	Delete the registered information of the ghost nodes for which the connection information cannot be checked

rosnode list: Check the list of running nodes

This is a command to list up all nodes connected to ‘roscore’. If you have only run ‘roscore’ and the previous example(‘turtlesim_node’, ‘turtle_teleop_key’), then you will see that ‘rosout’, which is executed along with ‘roscore’ for recording logs, and ‘teleop_turtle’ and ‘turtlesim’ nodes that were executed in the previous example will be on the list.

```
$ rosnode list
/rosout
/teleop_turtle
/turtlesim
```

Running a node and the actual node name

In the previous example, when ‘turtlesim_node’ and ‘turtle_teleop_key’ are being executed and yet ‘teleop_turtle’ and ‘turtlesim’ are appearing in the ‘rosnode list’ is because the name of the running node is different from the actual node name. For example, the ‘turtle_teleop_key’ node is configured as “ros :: init (argc, argv, “teleop_turtle”);” in the source file. We recommend creating the same name of the running node and the actual node name.

rosnode ping [NODE_NAME]: Test connection with a specific node

The following is a test to check whether the turtlesim node is actually connected to the computer that is currently used. If it is connected, it will receive an XMLRPC response from the corresponding node as follows.

```
$ rosnode ping /turtlesim
rosnode: node is [/turtlesim]
pinging /turtlesim with a timeout of 3.0s
xmlrpc reply from http://192.168.1.100:45470/       time=0.377178ms
```

If there is a problem running the corresponding node or the communication has been interrupted, the following error message will appear.

```
ERROR: connection refused to [http://192.168.1.100:55996/]
```

rosnode info [NODE_NAME]: Check information of a specific node

By using the 'rosnode info' command, we can check the information of a specific node. Generally, you can check Publications, Subscriptions, Services as well as the information about the running node URI and topic input/output. A lot of information that is displayed has been omitted here, so it is recommended to run this practice.

```
$ rosnode info /turtlesim
------------------------------------------------
Node [/turtlesim] Publications:
 * /turtle1/color_sensor [turtlesim/Color]
~ omitted ~
```

rosnode machine [PC_NAME OR IP]: Check the list of nodes running on the corresponding PC

By using this command, we can see the list of nodes that are running on a specific device (PC or mobile device).

```
$ rosnode machine 192.168.1.100
/rosout
/teleop_turtle
/turtlesim
```

rosnode kill [NODE_NAME]: Stop running a specific node

This is a command to close a running node. We can close a node directly using [Ctrl+c] on the terminal window where the node was running, but we can also close it by specifying the node to kill as follows.

```
$ rosnode kill /turtlesim
killing /turtlesim
killed
```

If we close a node with this command, a warning message will appear on the terminal window where the corresponding node is running as shown below and the node will be closed.

```
[WARN] [1512635717.915684117]: Shutdown request received.
[WARN] [1512635717.915711940]: Reason given for shutdown: [user request]
```

rosnode cleanup: Delete the registered information of the ghost nodes with unverified connection information

This command deletes unverified connection information of ghost nodes. When a node shuts down abnormally due to an unexpected error, this command deletes the corresponding node from the list that the connection information has been cut. Although this command is not frequently used, it is a useful command because you do not need to terminate and run 'roscore' again to delete up the ghost node.

```
$ rosnode cleanup
```

5.4.3. rostopic: ROS Topic

Understanding topics is necessary, so please refer to the Section 4.1 ROS Terminology.

Command	Description
rostopic list	Show the list of active topics
rostopic echo [TOPIC_NAME]	Show the content of a message in real-time for a specific topic
rostopic find [TYPE_NAME]	Show the topics that use specific message type
rostopic type [TOPIC_NAME]	Show the message type of a specific topic
rostopic bw [TOPIC_NAME]	Show the message data bandwidth of a specific topic
rostopic hz [TOPIC_NAME]	Show the message data publishing period of a specific topic

Command	Description
rostopic info [TOPIC_NAME]	Show the information of a specific topic
rostopic pub [TOPIC_NAME] [MESSAGE_TYPE] [PARAMETER]	Publish a message with the specific topic name

Close all the nodes before running the example regarding ROS topic. Then run 'roscore', 'turtlesim_node' and 'turtle_teleop_key' in three different terminal windows by running the following commands.

```
$ roscore
$ rosrun turtlesim turtlesim_node
$ rosrun turtlesim turtle_teleop_key
```

rostopic list: Show the list of active topics

The 'rostopic' list command lists up the topics that are currently being sent and received.

```
$ rostopic list
/rosout
/rosout_agg
/turtle1/cmd_vel
/turtle1/color_sensor
/turtle1/pose
```

If you add the '-v' option to the 'rostopic list' command, it separates the published topics and the subscribed topics, and shows the message type for each topic as well.

```
$ 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
```

rostopic echo [TOPIC_NAME]: Show the content of a message in real-time for a specific topic

The following example shows the data for 'x', 'y', 'theta, linear_velocity', and 'angular_velocity' that constitutes the '/turtle1/pose' topic in real-time.

```
$ rostopic echo /turtle1/pose
x: 5.35244464874
y: 5.544444561
theta: 0.0
linear_velocity: 0.0
angular_velocity: 0.0
~ omitted ~
```

rostopic find [TYPE_NAME]: Show the topics that use a specific message type

```
$ rostopic find turtlesim/Pose
/turtle1/pose
```

rostopic type [TOPIC_NAME]: Show the message type of a specific topic

```
$ rostopic type /turtle1/pose
turtlesim/Pose
```

rostopic bw [TOPIC_NAME]: Show the message data bandwidth of a specific topic

In the following example, we can see that the data bandwidth used in the '/turtle1/pose' topic is 1.27KB per second on average.

```
$ rostopic bw /turtle1/pose
subscribed to [/turtle1/pose]
average: 1.27KB/s
mean: 0.02KB min: 0.02KB max: 0.02KB window: 62 ...
~ omitted ~
```

rostopic hz [TOPIC_NAME]: Show the message data publishing period of a specified topic

In the following example, we can check the publishing period of the '/turtle1/pose' data. As a result, we can see that the message is being published at a period of about 62.5Hz (0.016sec = 16msec).

```
$ rostopic hz /turtle1/pose
subscribed to [/turtle1/pose]
average rate: 62.502
min: 0.016s max: 0.016s std dev: 0.00005s window: 62
```

rostopic info [TOPIC_NAME]: Show the information of a specific topic

In the following example, we can see that the '/turtle1/pose' topic uses the 'turtlesim/Pose' message type and published from the '/turtlesim' node, and there are no topics that are being subscribed.

```
$ rostopic info /turtle1/pose
Type: turtlesim/Pose
Publishers:
 * /turtlesim (http://192.168.1.100:42443/)
Subscribers: None
```

rostopic pub [TOPIC_NAME] [MESSAGE_TYPE] [PARAMETER]: Publish a message with the specified topic name

The following is an example of publishing a message with the topic name '/turtle1/cmd_vel' with a message type of 'geometry_msgs/Twist'.

```
$ 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
```

The following is a description for each of the options.

- -1 Publish the message only once (it runs only once, but it runs for 3 seconds as shown above).
- /turtle1/cmd_vel the specific topic name
- geometry_msgs/Twist the name of the message type published
- -- '[2.0, 0.0, 0.0]' '[0.0, 0.0, 1.8]' moving in the x-axis coordinate with a speed of 2.0m per second, and with a rotation of 1.8rad per second about the z-axis

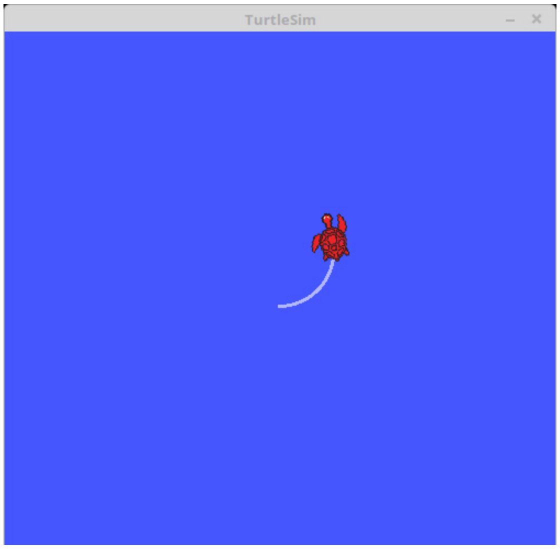

FIGURE 5-2 Screen showing the published message applied

5.4.4. rosservice: ROS Service

Understanding services in necessary, so please refer to the Section 4.1 ROS Terminology.

Command	Description
rosservice list	Display information of active services
rosservice info [SERVICE_NAME]	Display information of a specific service
rosservice type [SERVICE_NAME]	Display service type
rosservice find [SERVICE_TYPE]	Search services with a specific service type
rosservice uri [SERVICE_NAME]	Display the ROSRPC URI service
rosservice args [SERVICE_NAME]	Display the service parameters
rosservice call [SERVICE_NAME] [PARAMETER]	Request service with the input parameter

Close all nodes before running the example regarding ROS service. Then run 'roscore', 'turtlesim_node' and 'turtle_teleop_key' in different terminal windows by running the following commands.

```
$ roscore
$ rosrun turtlesim turtlesim_node
$ rosrun turtlesim turtle_teleop_key
```

rosservice list: Display information of active services

This command displays information about the active services. All services that are in use in the same network will be displayed.

```
$ 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
```

rosservice info [SERVICE_NAME]: Display information of a specific service

The following is an example of checking the node name, URI, type, and parameter of the '/ turtle1/set_pen' service using the info option of 'rosservice'.

```
$ rosservice info /turtle1/set_pen
Node: /turtlesim
URI: rosrpc://192.168.1.100:34715
Type: turtlesim/SetPen
Args: r g b width off
```

rosservice type [SERVICE_NAME]: Display service type

In the following example, we can see that the '/turtle1/set_pen' service is the type of 'turtlesim/ SetPen'.

```
$ rosservice type /turtle1/set_pen
turtlesim/SetPen
```

rosservice find [SERVICE_TYPE]: Search services with a specific service type

The following example is a command to search for services with the type 'turtlesim/SetPen'. We can see that the result is '/turtle1/set_pen'.

```
$ rosservice find turtlesim/SetPen
/turtle1/set_pen
```

rosservice uri [SERVICE_NAME]: Display the ROSRPC URI service

By using the uri option of 'rosservice', we can check the ROSRPC URI of the '/turtle1/set_pen' service as shown below.

```
$ rosservice uri /turtle1/set_pen
rosrpc://192.168.1.100:50624
```

rosservice args [SERVICE_NAME]: Display the service parameters

Let us check each parameter of the '/turtle1/set_pen' service as shown in the following example. Through this command, we can check that the parameters being used in the '/turtle1/set_pen' service are 'r', 'g', 'b', 'width', and 'off'.

```
$ rosservice args /turtle1/set_pen
r g b width off
```

rosservice call [SERViCE_NAME] [PARAMETER]: Request service with the input parameter

The following example is a command that requests the '/turtle1/set_pen' service. The values '255 0 0 5 0' correspond to the parameters (r, g, b, width, off) used for the '/turtle1/set_pen' service. The value of 'r' which represents red has the maximum value of 255 while 'g' and 'b' are both '0', so the color of the pen will be red. The 'width' is set to a thickness of 5, and 'off' is set to 0 (false), so the line will be displayed. 'rosservice call' is an extremely useful command that is used for testing when using a service, and is frequently used.

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

Using the command above, we requested for a service that changes the properties of the pen used in turtlesim, and by ordering a command to move in 'turtle_teleop_key', we can see that the color of pen that was white is now displayed in red as below.

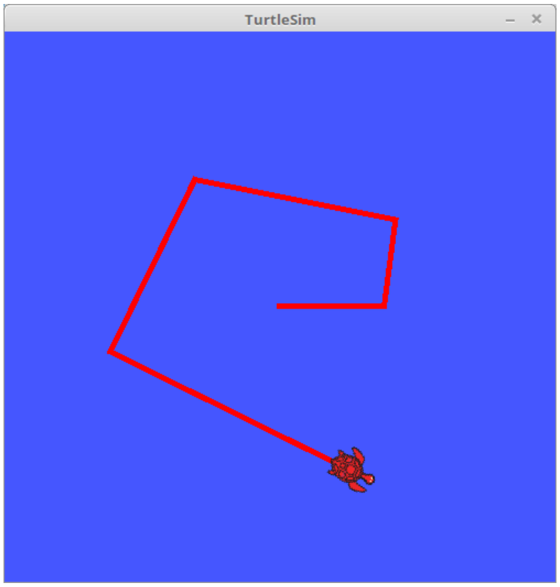

FIGURE 5-3 Example of rosservice call

5.4.5. rosparam: ROS Parameter

Understanding parameters is necessary, so please refer to the Section 4.1 ROS Terminology.

Command	Description
rosparam list	View parameter list
rosparam get [PARAMETER_NAME]	Get parameter value
rosparam set [PARAMETER_NAME]	Set parameter value
rosparam dump [FILE_NAME]	Save parameter to a specific file
rosparam load [FILE_NAME]	Load parameter that is saved in a specific file
rosparam delete [PARAMETER_NAME]	Delete parameter

Let us close all the nodes before running the example regarding ROS parameter. Then run 'roscore', 'turtlesim_node' and 'turtle_teleop_key' in different terminal windows by running the following commands.

```
$ roscore
$ rosrun turtlesim turtlesim_node
$ rosrun turtlesim turtle_teleop_key
```

rosparam list: View parameter list

A list of parameters being used in the same network will be shown.

```
$ rosparam list
/background_b
/background_g
/background_r
/rosdistro
/roslaunch/uris/host_192_168_1_100__39536
/rosversion
/run_id
```

rosparam get [PARAMETER_NAME]: Get parameter value

If you want to check the value of a specific parameter, you can type in the parameter name as an option after the 'rosparam get' command.

```
$ rosparam get /background_b
255
```

If you want to check the values of all the other parameters apart from a specific parameter, you can use '/' as an option which will show the values of all the parameters as below.

```
$ rosparam get /
background_b: 255
background_g: 86
background_r: 69
rosdistro: 'kinetic'
roslaunch:
  uris: {host_192_168_1_100__43517: 'http:// 192.168.1.100:43517/'}
rosversion: '1.12.7'
run_id: c2d0b528-6536-11e7-935b-08d40c80c500
```

rosparam dump [FILE_NAME]: Save parameter to a specific file

The following example is a command that saves the current parameter value to the 'parameters.yaml' file. This is very useful because it can save a parameter value that was used to apply in the next run ('~/' represents the user's home directory).

```
$ rosparam dump ~/parameters.yaml
```

rosparam set [PARAMETER_NAME]: Set parameter value

The following example sets the 'background_b' parameter of the turtlesim node, which is a parameter regarding the background color, to '0'.

```
$ rosparam set background_b 0
$ rosservice call clear
```

RGB is changed from '255, 86, 69' to '0, 86, 69', so the color becomes a dark green as shown in the picture to the right in Image 5-4. However, the 'turtlesim' node does not read and apply the parameters right away, so we need to first modify the parameters with the command 'rosparam set background_b 0' and then refresh the screen with the command 'rosservice call clear'. The application of a parameter changes according to the node.

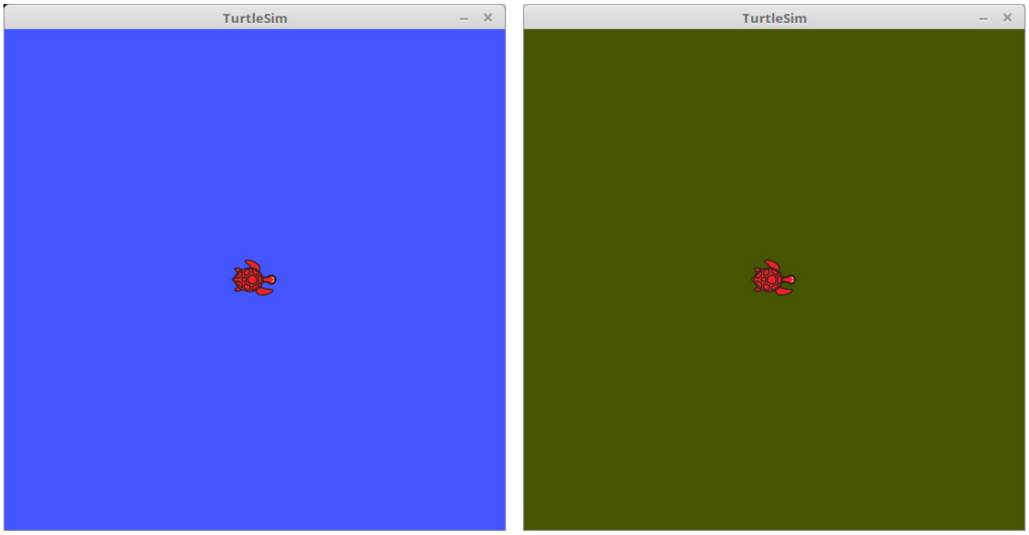

FIGURE 5-4 Example of 'rosparam set'

rosparam load [FILE_NAME]: Load parameter that is saved in a specified file

Contrary to 'rosparam dump', this loads the 'parameters.yaml' file and uses it as the current parameter value. As shown in the following example, if we run the 'rosservice call clear' command, it will be replaced with the parameter value of the loaded file, and the background color that was changed to green in Image 5-4 will change back to blue as it was when the dump command was run. Rosparam load is a useful command that is used quite often, so let us become familiar with it.

```
$ rosparam load ~/parameters.yaml
$ rosservice call clear
```

rosparam delete [PARAMETER_NAME]: Delete parameter

This is the command to delete a specific parameter.

```
$ rosparam delete /background_b
```

5.4.6. rosmsg: ROS Message Information

Understand messages is necessary, so please refer to the Section 4.1 ROS Terminology.

Command	Description
rosmsg list	Show a list of all messages
rosmsg show [MESSAGE_NAME]	Show information of a specified message
rosmsg md5 [MESSAGE_NAME]	Show the md5sum
rosmsg package [PACKAGE_NAME]	Show a list of messages used in a specified package
rosmsg packages	Show a list of all packages that use messages

Let us close all the nodes before running the example regarding ROS message information. Then run ‘roscore’, ‘turtlesim_node’ and ‘turtle_teleop_key’ in different terminal windows by running the following commands.

```
$ roscore
$ rosrun turtlesim turtlesim_node
$ rosrun turtlesim turtle_teleop_key
```

rosmsg list: Show a list of all messages

This command lists all of the messages in the packages currently installed. The resulting value may vary depending on the packages included in ROS.

```
$ rosmsg list
actionlib/TestAction
actionlib/TestActionFeedback
actionlib/TestActionGoal
actionlib/TestActionResult
```

```
actionlib/TestFeedback
actionlib/TestGoal
sensor_msgs/Joy
sensor_msgs/JoyFeedback
sensor_msgs/JoyFeedbackArray
sensor_msgs/LaserEcho
zeroconf_msgs/DiscoveredService
~ omitted ~
```

rosmsg show [MESSAGE_NAME]: Show information of a specific message

This shows information of a specific message. The following is an example of displaying the 'turtlesim/Pose' message information. We can see the message contains five pieces of information of the float32 type variables 'x', 'y', 'theta', 'linear_velocity', and 'angular_velocity'.

```
$ rosmsg show turtlesim/Pose
float32 x
float32 y
float32 theta
float32 linear_velocity
float32 angular_velocity
```

rosmsg md5 [MESSAGE_NAME]: Show the md5sum

The following is an example to check the md5 information of the 'turtlesim/Pose' message. When an MD5 problem occurs during message communication, you will need to check the md5sum with this command. Generally it is not commonly used. For more information on md5sum, refer to Section 4.1 ROS Terminology.

```
$ rosmsg md5 turtlesim/Pose
863b248d5016ca62ea2e895ae5265cf9
```

rosmsg package [PACKAGE_NAME]: Show a list of messages used in a specific package

By using this command, we can see the messages used in a specific package.

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

rosmsg packages: Show the list of all packages that use messages

```
$ rosmsg packages
actionlib
actionlib_msgs
actionlib_tutorials
base_local_planner
bond
control_msgs
costmap_2d
~omitted~
```

5.4.7. rossrv: ROS Service Information

Understanding services is essential, so please refer to the Section 4.1 ROS Terminology.

Command	Description
rossrv list	Show a list of all services
rossrv show [SERVICE_NAME]	Show information of a specific service
rossrv md5 [SERVICE_NAME]	Show the md5sum
rossrv package [PACKAGE_NAME]	Show a list of services used in a specific package
rossrv packages	Show a list of all packages that use services

Let us close all the nodes before running the example regarding ROS service information. Then run 'roscore', 'turtlesim_node' and 'turtle_teleop_key' in different terminal windows by running the following commands.

```
$ roscore
$ rosrun turtlesim turtlesim_node
$ rosrun turtlesim turtle_teleop_key
```

rossrv list: Show a list of all services

This is a command to list all of the services in the packages currently installed on ROS. Depending on the packages currently included in ROS, the resulting value may vary.

```
$ rossrv list
control_msgs/QueryCalibrationState
control_msgs/QueryTrajectoryState
diagnostic_msgs/SelfTest
dynamic_reconfigure/Reconfigure
gazebo_msgs/ApplyBodyWrench
gazebo_msgs/ApplyJointEffort
gazebo_msgs/BodyRequest
gazebo_msgs/DeleteModel
~ omitted ~
```

rossrv show [SERVICE_NAME]: Show information of a specific service

The following is an example of displaying the 'turtlesim/SetPen' service information. We can see that it is a service containing five pieces of information of uint8 type variables 'r', 'g', 'b', 'width', and 'off'. For your information, '---' is used as a line separating the request and response in the service file, so for the case of 'turtlesim/SetPen' we can see that there is only a request and no content that corresponds to a response. For more information on service files, refer to Section 4.3 and practical examples are available in Section 7.3.

```
$ rossrv show turtlesim/SetPen
uint8 r
uint8 g
uint8 b
uint8 width
uint8 off
---
```

rossrv md5 [SERVICE_NAME]: Show the md5sum

The following is an example to check the md5 information of the 'turtlesim/SetPen' service. When an MD5 problem occurs during the service request/response, you will need to check the md5sum, and this is the command you can use. Generally it is not commonly used.

```
$ rossrv md5 turtlesim/SetPen
9f452acce566bf0c0954594f69a8e41b
```

rossrv package [PACKAGE_NAME]: Show a list of services used in a specific package

This command lists up the services used in a specific package.

```
$ rossrv package turtlesim
turtlesim/Kill
turtlesim/SetPen
turtlesim/Spawn
turtlesim/TeleportAbsolute
turtlesim/TeleportRelative
```

rossrv packages: Show a list of all packages that use services

```
$ rossrv packages
control_msgs
diagnostic_msgs
dynamic_reconfigure
gazebo_msgs
map_msgs
nav_msgs
navfn nodelet
oroca_ros_tutorials
roscpp
sensor_msgs
std_srvs
tf
tf2_msgs
turtlesim
~omitted~
```

5.4.8. rosbag: ROS Log Information

It was explained in Section 4.1 ROS Terminology that in ROS we can save various messages in bag format and play them back when necessary in order to reproduce the same environment when data is recorded. Rosbag is a program that creates, plays, and compresses bags, and has the following various functions.

Command	Description
rosbag record [OPTION] [TOPIC_NAME]	Record the message of a specific topic on the bsg file
rosbag info [FILE_NAME]	Check information of a bag file
rosbag play [FILE_NAME]	Play a specific bag file
rosbag compress [FILE_NAME]	Compress a specific bag file
rosbag decompress [FILE_NAME]	Decompresses a specific bag file
rosbag filter [INPUT_FILE] [OUTPUT_FILE] [OPTION]	Create a new bag file with the specific content removed
rosbag reindex bag [FILE_NAME]	Reindex
rosbag check bag [FILE_NAME]	Check if the specific bag file can be played in the current system
rosbag fix [INPUT_FILE] [OUTPUT_FILE] [OPTION]	Fix the bag file version that was saved as an incompatible version

Let us close all the nodes before running the example regarding ROS log information. Then run 'roscore', 'turtlesim_node' and 'turtle_teleop_key' in different terminal windows by running the following commands.

```
$ roscore
$ rosrun turtlesim turtlesim_node
$ rosrun turtlesim turtle_teleop_key
```

rosbag record [OPTION][TOPIC_NAME]: Record the message of a specific topic

First, we will use the rostopic list command to check the list of topics currently being used in the ROS network.

```
$ rostopic list
/rosout
/rosout_agg
/turtle1/cmd_vel
/turtle1/color_sensor
/turtle1/pose
```

As shown in the following example, from the topics that are in use, we will type in the topic we want to record as an option when start recording the bag file. After we start recording, on the terminal window that the 'turtle_teleop_key' node is running, if we control the turtle by using the arrow keys on the keyboard, the '/turtle1/cmd_vel' topic assigned as the option will be recorded. Then if we press [Ctrl+c] to stop recording, a bag file '2017-07-10-14-16-28.bag' will be created as shown below.

```
$ rosbag record /turtle1/cmd_vel
[INFO] [1499663788.499650818]: Subscribing to /turtle1/cmd_vel
[INFO] [1499663788.502937962]: Recording to 2017-07-10-14-16-28.bag.
```

If you wish to record all the topics at the same time, then add the '-a' option.

```
$ rosbag record -a
[WARN] [1499664121.243116836]: --max-splits is ignored without --split
[INFO] [1499664121.248582681]: Recording to 2017-07-10-14-22-01.bag.
[INFO] [1499664121.248879947]: Subscribing to /turtle1/color_sensor
[INFO] [1499664121.252689657]: Subscribing to /rosout
[INFO] [1499664121.257219911]: Subscribing to /rosout_agg
[INFO] [1499664121.260671283]: Subscribing to /turtle1/pose
```

rosbag info [bag FILE_NAME]: Check information of a bag file

By using this command we can check the information of a bag file. The following example is a recorded '/turtle1/cmd_vel' topic, and 373 messages were recorded. The message type used is 'geometry_msgs/Twist', and we can also check information such as path, bag version, time, etc.

```
$ rosbag info 2017-07-10-14-16-28.bag
path:          2017-07-10-14-16-28.bag
version:       2.0
duration:      17.4s
start:         Jul 10 2017 14:16:30.36 (1499663790.36)
end:           Jul 10 2017 14:16:47.78 (1499663807.78)
size:          44.5 KB
messages:      373
compression:   none [1/1 chunks]
types:         geometry_msgs/Twist [9f195f881246fdfa2798d1d3eebca84a]
topics:        /turtle1/cmd_vel   373 msgs    : geometry_msgs/Twist
```

rosbag play [bag FILE_NAME]: Play a specific bag file

The following example is a command to play the recorded '2017-07-10-14-16-28.bag' file. The message '/turtle1/cmd_vel' from the time of the recording is transmitted exactly, and we can see the turtle move in the screen. However, the same result as shown in Figure 5-5 can only be obtained when 'turtlesim_node' is restarted and initialize the robot trajectory and robot position.

```
$ rosbag play 2017-07-10-14-16-28.bag
[INFO] [1499664453.406867251]: Opening 2017-07-10-14-16-28.bag
Waiting 0.2 seconds after advertising topics... done.
Hit space to toggle paused, or 's' to step.
[RUNNING]  Bag Time: 1499663790.357031   Duration: 0.000000 / 17.419737
[RUNNING]  Bag Time: 1499663790.357031   Duration: 0.000000 / 17.419737
[RUNNING]  Bag Time: 1499663790.357163   Duration: 0.000132 / 17.419737
~ omitted ~
```

As in the following figure, we can see that the original data and the data during playback are the same.

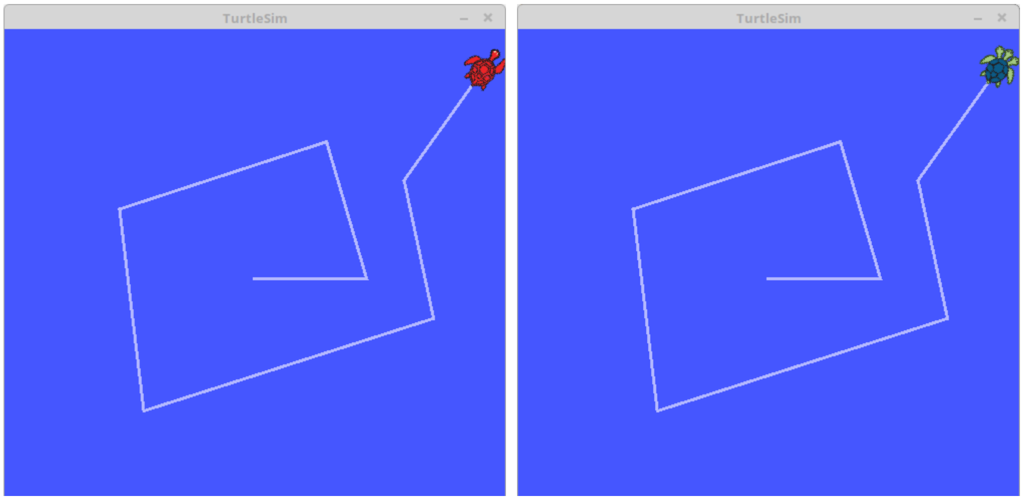

FIGURE 5-5 Example of rosbag play

rosbag compress [bag FILE_NAME]: Compress a specific bag file

A bag file recorded for a short period of time creates relatively small size file, which is not a problem, but if a bag file records data for a long period of time then it takes up a lot of storage space. The following compression command is used in this case, and by compressing it will take up very little storage space.

```
$ rosbag compress 2017-07-10-14-16-28.bag
2017-07-10-14-16-28.bag    0%      0.0 KB 00:00
2017-07-10-14-16-28.bag    100%    35.0 KB 00:00
```

The bag file from the example above is reduced to a quarter as shown below. And the original file before compression is separately saved with 'orig' tag added to its name.

```
2017-07-10-14-16-28.bag 12.7kB
2017-07-10-14-16-28.orig.bag 45.5kB
```

rosbag decompress [bag FILE_NAME]: Decompresses a specific bag file

To decompress the compressed bag file, use the following command. This will restore the bag file to the original state before compression.

```
$ rosbag decompress 2017-07-10-14-16-28.bag
2017-07-10-14-16-28.bag    0%      0.0 KB 00:00
2017-07-10-14-16-28.bag    100%    35.0 KB 00:00
```

5.5. ROS Catkin Commands

ROS Catkin commands are used when building a package using the catkin build system.

Command	Importance	Description
catkin_create_pkg	★★★	Automatic creation of package
catkin_make	★★★	Build based on catkin build system
catkin_eclipse	★★☆	Modify package created by catkin build system so that it can be used in Eclipse
catkin_prepare_release	★★☆	Cleanup log and tag version during release
catkin_generate_changelog	★★☆	Create or update 'CHANGELOG.rst' file during release
catkin_init_workspace	★★☆	Initialize workspace of the catkin build system
catkin_find	★☆☆	Search catkin

catkin_create_pkg: Automatic creation of package

```
catkin_create_pkg [PACKAGE_NAME] [DEPENDENCY_PACKAGE1] [DEPENDENCY_PACKAGE 2] ...
```

The ‘catkin_create_pkg’ is a command that creates an empty package containing ‘CMakeLists.txt’ and ‘package.xml’ files. For detailed instructions, please refer to Section 4.9 where the build system of ROS is explained. The following is an example of using ‘catkin_create_pkg’ command to create ‘my_package’ package which depends on ‘roscpp’ and ‘std_msgs’.

```
$ catkin_create_pkg my_package roscpp std_msgs
```

catkin_make: Build based on catkin build system

```
catkin_make [OPTION]
```

The ‘catkin_make’ is a command to build a package created by a user or a downloaded package. The following is an example of building all packages in the ‘~/catkin_ws/src’ folder.

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

To build just some of the packages and not all of the packages, run with the ‘--pkg [PACKAGE_NAME]’ option as shown below.

```
$ catkin_make --pkg user_ros_tutorials
```

catkin_eclipse: Modify package created by catkin build system so that it can be used in Eclipse

The ‘catkin_eclipse’ is a command to configure an environment of a package for managing and programming with Eclipse, one of the Integrated Development Environments (IDEs). Running this command will create project files for Eclipse such as ‘~/catkin_ws/build/.cproject’, ‘~/catkin_ws/build/.project’, etc. From the Eclipse menu, by selecting [Makefile Project with Existing Code] and choosing ‘~/catkin_ws/build/’ we can manage all packages in ‘~/catkin_ws/src’ from Eclipse.

```
$ cd ~/catkin_ws
$ catkin_eclipse
```

catkin_generate_changelog: Create file CHANGELOG.rst

The ‘catkin_generate_changelog’ is a command that creates the ‘CHANGELOG.rst’ file that logs the changes when updating the version of a package.

catkin_prepare_release: Manage change records and version tags when preparing for release

The ‘catkin_prepare_release’ is a command used to update the ‘CHANGELOG.rst’ created by the ‘catkin_generate_changelog’ command. The ‘catkin_generate_changelog’ and ‘catkin_prepare_release’ commands are used when registering a created package with the official ROS repository, or when updating the version of a registered package.

catkin_init_workspace: Initialize working folder of the catkin build system

The ‘catkin_init_workspace’ is a command to initialize the user working folder (~/catkin_ws/src). As mentioned in Section 3.1, except for special occasions, this command is executed only once during the ROS installation.

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

catkin_find: Search Catkin, find and show the workspace

The ‘catkin_find’ is a command that shows the working folders for each project.

```
catkin_find [PACKAGE_NAME]
```

By using the ‘catkin_find’ command, we can find out all the working folders we are using. Additionally, if we run ‘catkin_find PACKAGE_NAME’, it will show the working folders relevant to the package specified in the option as shown below.

```
$ catkin_find
/home/pyo/catkin_ws/devel/include
/home/pyo/catkin_ws/devel/lib
/home/pyo/catkin_ws/devel/share
/opt/ros/kinetic/bin
/opt/ros/kinetic/etc
/opt/ros/kinetic/include
/opt/ros/kinetic/lib
/opt/ros/kinetic/share
```

```
$ catkin_find turtlesim
/opt/ros/kinetic/include/turtlesim
/opt/ros/kinetic/lib/turtlesim
/opt/ros/kinetic/share/turtlesim
```

5.6. ROS Package Commands

ROS package commands are used to manage ROS packages, such as showing information of packages and installing related packages.

Command	Importance	Command Explanation	Description
ropack	★★★	ros+pack(package)	View information regarding a specific ROS package
rosinstall	★★☆	ros+install	Install additional ROS packages
rosdep	★★☆	ros+dep(dependencies)	Install dependency package of the ROS corresponding package
roslocate	☆☆☆	ros+locate	Show information of ROS package
roscreate-pkg	☆☆☆	ros+create-pkg(package)	Automatic creation of ROS package (used in previous rosbuild system)
rosmake	☆☆☆	ros+make	Build ROS package (used in previous rosbuild system)

rospack: View information regarding a specific ROS package

```
rospack [OPTION] [PACKAGE_NAME]
```

The ‘rospack’ is a command to show information such as the save location, dependency, entire package list regarding the specific ROS package, and we can use options such as ‘find’, ‘list’, ‘depends-on’, ‘depends’, ‘profile’, etc. As shown in the example below, if we specify a package name after the ‘rospack find’ command, the saved location of the package will be shown.

```
$ rospack find turtlesim
/opt/ros/kinetic/share/turtlesim
```

The ‘rospack list’ command shows all the packages in the PC. By combining the ‘rospack list’ command with the Linux search command ‘grep’ we can easily find a package. For instance, running ‘rospack list | grep turtle’ will only display the packages related to turtle as shown in the example below.

```
$ rospack list
actionlib /opt/ros/kinetic/share/actionlib
actionlib_msgs /opt/ros/kinetic/share/actionlib_msgs
actionlib_tutorials /opt/ros/kinetic/share/actionlib_tutorials
amcl /opt/ros/kinetic/share/amcl
angles /opt/ros/kinetic/share/angles
base_local_planner /opt/ros/kinetic/share/base_local_planner
bfl /opt/ros/kinetic/share/bfl
```

```
$ 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
turtlesim /opt/ros/kinetic/share/turtlesim
```

If we specify a package name after the ‘rospack depends-on’ command, it will only show the packages that are using the specific package as shown in the following example.

```
$ rospack depends-on turtlesim
turtle_tf2
turtle_tf
turtle_actionlib
```

If we specify the package name after the ‘rospack depends’ command, it will show the dependency packages needed to run the specific package as shown in the following example.

```
$ rospack depends turtlesim
cpp_common
rostime
roscpp_traits
roscpp_serialization
genmsg
genpy
message_runtime
std_msgs
geometry_msgs
catkin
gencpp
genlisp
message_generation
```

```
rosbuild
rosconsole
rosgraph_msgs
xmlrpcpp
roscpp
rospack
roslib
std_srvs
```

The ‘rospack profile’ command re-indexes the package by checking the package information and working folders such as ‘/opt/ros/kinetic/share’ or ‘~/catkin_ws/src’ where packages are saved. You can use this command when a newly added package is not listed by the ‘roscd’ command.

```
$ rospack profile
Full tree crawl took 0.021790 seconds.
Directories marked with (*) contain no manifest.  You may
want to delete these directories.
To get just of list of directories without manifests,
re-run the profile with --zombie-only
-------------------------------------------------------------
0.020444   /opt/ros/kinetic/share
0.000676   /home/pyo/catkin_ws/src
0.000606   /home/pyo/catkin_ws/src/ros_tutorials
0.000240 * /opt/ros/kinetic/share/OpenCV-3.2.0-dev
0.000054 * /opt/ros/kinetic/share/OpenCV-3.2.0-dev/haarcascades
0.000035 * /opt/ros/kinetic/share/doc
0.000020 * /opt/ros/kinetic/share/OpenCV-3.2.0-dev/lbpcascades
0.000008 * /opt/ros/kinetic/share/doc/liborocos-kdl
```

rosinstall: Install additional ROS packages

Th ‘rosinstall’ is a command that automatically installs or updates ROS packages managed by Source Code Managements (SCMs) such as SVN, Mercurial, Git, Bazaar. As seen in Section 3.1, once we run the program, necessary packages will be automatically installed or updated whenever there are package updates.

rosdep: Install dependency package of the ROS corresponding package

```
rosdep [OPTION]
```

The ‘rosdep’ is a command that installs the dependency file of the specific package. There are options such as ‘check’, ‘install’, ‘init’ and ‘update’. As shown in the following example, running ‘rosdep check PACKAGE_NAME’ will check the dependency of the specific package. Running ‘rosdep install PACKAGE_NAME’ will install the dependency package of the specific package. Furthermore, there are also ‘rosdep init’ or ‘rosdep update’, but please refer to Section 3.1 for the detailed instructions.

```
$ rosdep check turtlesim
All system dependencies have been satisified
$ rosdep install turtlesim
All required rosdeps installed successfully
```

roslocate: Show information of ROS package

```
roslocate [OPTION] [PACKAGE_NAME]
```

The ‘roslocate’ is a command that shows information such as the ROS version used for the package, SCM type, repository location, and so on. Available options are ‘info’, ‘vcs’, ‘type’, ‘uri’, ‘repo’, etc. Here we will look at ‘info’, which shows all of this information at once.

```
$ roslocate info turtlesim
Using ROS_DISTRO: kinetic
- git:
local-name: turtlesim
uri: https://github.com/ros/ros_tutorials.git
version: kinetic-devel
```

roscreate-pkg: Automatic creation of ROS package (used in previous rosbuild system)

The ‘roscreate-pkg’ is a command that automatically creates a package similar to the ‘catkin_create_pkg’ command. It is a command that was used in the previous rosbuild system, before the catkin build system. It has been left for version compatibility, and is not used in the recent versions.

rosmake: Build ROS package (used in previous rosbuild system)

The ‘rosmake’ is a command that builds a package similar to the ‘catkin_make’ command. It is a command that was used in the previous rosbuild system, before the catkin build system. It has been left for version compatibility, and is not used in the recent versions.