ROS Kinetic Cheatsheet

Filesystem Management Tools

rospack	A tool for inspecting packages.
rospack profile	Fixes path and pluginlib problems.
roscd	Change directory to a package.
rospd/rosd	Pushd equivalent for ROS.
rosls	Lists package or stack information.
rosed	Open requested ROS file in a text editor.
roscp	Copy a file from one place to another.
rosdep	Installs package system dependencies.
roswtf	Displays a errors and warnings about a running ROS system or launch file.
catkin_create_pkg	Creates a new ROS stack.
wstool	Manage many repos in workspace.
catkin_make	Builds a ROS catkin workspace.
rqt_dep	Displays package structure and dependencies.

Usage:

```
$ rospack find [package]
$ roscd [package[/subdir]]
$ rospd [package[/subdir] | +N | -N]
$ rosd
$ rosls [package[/subdir]]
$ rosed [package] [file]
$ roscp [package] [file] [destination]
$ rosdep install [package]
$ roswtf or roswtf [file]
$ catkin_create_pkg [package_name] [depend1]..[dependN]
$ wstool [init | set | update]
$ catkin_make
$ rqt_dep [options]
```

Start-up and Process Launch Tools

roscore

The basis nodes and programs for ROS-based systems. A roscore must be running for ROS nodes to communicate.

Usage:

```
$ roscore
```

rosrun

Runs a ROS package's executable with minimal typing.

Usage:

```
$ rosrun package_name executable_name
```

Example (runs turtlesim):

```
$ rosrun turtlesim turtlesim_node
```

roslaunch

Starts a roscore (if needed), local nodes, remote nodes via SSH, and sets parameter server parameters.

Examples:

Launch a file in a package:

```
$ roslaunch package_name file_name.launch
```

Launch on a different port:

```
$ roslaunch -p 1234 package_name file_name.launch
```

Launch on the local nodes:

```
$ roslaunch --local package_name file_name.launch
```

Introspection and Command Tools

rosnode

Displays debugging information about ROS nodes, including publications, subscriptions and connections.

Commands:

rosnode ping	Test connectivity to node.
rosnode list	List active nodes.
rosnode info	Print information about a node.
rosnode machine	List nodes running on a machine.
rosnode kill	Kill a running node.

Examples:

Display information:

```
$ rosnode info /name
```

List nodes on a machine:

```
$ rosnode machine aqy.local
```

Kill all nodes:

```
$ rosnode kill -a
```

Ping all nodes:

```
$ rosnode ping --all
```

rostopic

A tool for displaying information about ROS topics, including publishers, subscribers, publishing rate, and messages.

Commands:

rostopic bw	Display bandwidth used by topic.
rostopic echo	Print messages to screen.
rostopic find	Find topics by type.
rostopic hz	Display publishing rate of topic.
rostopic info	Print information about an active topic.
rostopic list	List all published topics.
rostopic pub	Publish data to topic.
rostopic type	Print topic type.

Examples:

Publish hello at 10 Hz:

```
$ rostopic pub -r 10 /topic_name std_msgs/String hello
```

Clear the screen after each message is published:

```
$ rostopic echo -c /topic_name
```

Display messages that match a given Python expression:

```
$ rostopic echo --filter "m.data==’foo’" /topic_name
```

Pipe the output of rostopic to rosmsg to view the msg type:

```
$ rostopic type /topic_name | rosmsg show
```

rosservice

A tool for listing and querying ROS services.

Commands:

rosservice list	Print information about active services.
rosservice node	Print name of node providing a service.
rosservice call	Call the service with the given args.
rosservice args	List the arguments of a service.
rosservice type	Print the service type.
rosservice uri	Print the service ROSRPC uri.
rosservice find	Find services by service type.

Examples:

Call a service from the command-line:

```
$ rosservice call /add_two_ints 1 2
```

Pipe the output of rosservice to rossrv to view the srv type:

```
$ rosservice type add_two_ints | rossrv show
```

Display all services of a particular type:

```
$ rosservice find rospy_tutorials/AddTwoInts
```

rosconsole

Tool for configuring the logger level of ROS nodes.

Commands:

rosconsole get	Display level of a logger.
rosconsole list	List loggers for a node.
rosconsole set	Set level for a logger.

Examples:

Set debug logger level:

```
$ rosconsole talker ros.roscpp_tutorials debug
```

rosparam

A tool for getting and setting ROS parameters on the parameter server using YAML-encoded files.

Commands:

rosparam set	Set a parameter.
rosparam get	Get a parameter.
rosparam load	Load parameters from a file.
rosparam dump	Dump parameters to a file.
rosparam delete	Delete a parameter.
rosparam list	List parameter names.

Examples:

List all the parameters in a namespace:

```
$ rosparam list /namespace
```

Setting a list with one as a string, integer, and float:

```
$ rosparam set /foo "[’1’, 1, 1.0]"
```

Dump only the parameters in a specific namespace to file:

```
$ rosparam dump dump.yaml /namespace
```

rosmsg/rossrv

Displays Message/Service (msg/srv) data structure definitions.

Commands:

rosmsg show	Display the fields in the msg/srv.
rosmsg list	Display names of all msg/srv.
rosmsg md5	Display the msg/srv md5 sum.
rosmsg package	List all the msg/srv in a package.
rosmsg packages	List all packages containing the msg/srv.

Examples:

Display the Pose msg:

```
$ rosmsg show Pose
```

List the messages in the nav_msgs package:

```
$ rosmsg package nav_msgs
```

List the packages using sensor_msgs/CameraInfo:

```
$ rosmsg packages sensor_msgs/CameraInfo
```

Logging Tools

rosbag

A set of tools for recording and playing back of ROS topics.

Commands:

rosbag record	Record a bag file with specified topics.
rosbag play	Play content of one or more bag files.
rosbag compress	Compress one or more bag files.
rosbag decompress	Decompress one or more bag files.
rosbag filter	Filter the contents of the bag.

Examples:

Record select topics:

```
$ rosbag record topic1 topic2
```

Replay all messages without waiting:

```
$ rosbag play -a demo_log.bag
```

Replay several bag files at once:

```
$ rosbag play demo1.bag demo2.bag
```

tf_echo

A tool that prints the information about a particular transformation between a source_frame and a target_frame.

Usage:

```
$ rosrun tf tf_echo <source_frame> <target_frame>
```

Examples:

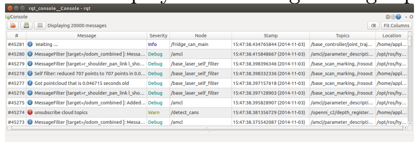
To echo the transform between /map and /odom:

```
$ rosrun tf tf_echo /map /odom
```

Logging Tools

rqt_console

A tool to display and filtering messages published on rosout.

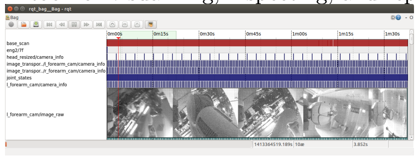


Usage:

```
$ rqt_console
```

rqt_bag

A tool for visualizing, inspecting, and replaying bag files.



Usage, viewing:

```
$ rqt_bag bag_file.bag
```

Usage, bagging:

```
$ rqt_bag *press the big red record button.*
```

rqt_logger_level

Change the logger level of ROS nodes. This will increase or decrease the information they log to the screen and rqt_console.
Usage: viewing

```
$ rqt_logger_level
```

Introspection & Command Tools

rqt_topic

A tool for viewing published topics in real time.
Usage:

```
$ rqt
Plugin Menu->Topic->Topic Monitor
```

rqt_msg, rqt_srv, and rqt_action

A tool for viewing available msgs, srvs, and actions.
Usage:

```
$ rqt
Plugin Menu->Topic->Message Type Browser
Plugin Menu->Service->Service Type Browser
Plugin Menu->Action->Action Type Browser
```

rqt_top

A tool for ROS specific process monitoring.
Usage:

```
$ rqt
Plugin Menu->Introspection->Process Monitor
```

rqt_publisher, and rqt_service_caller

Tools for publishing messages and calling services.
Usage:

```
$ rqt
Plugin Menu->Topic->Message Publisher
Plugin Menu->Service->Service Caller
```

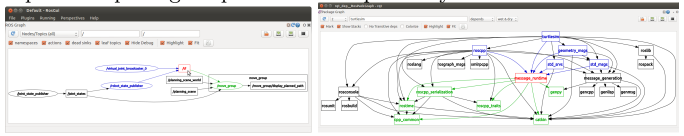
rqt_reconfigure

A tool for dynamically reconfiguring ROS parameters.
Usage:

```
$ rqt
Plugin Menu->Configuration->Dynamic Reconfigure
```

rqt_graph, and rqt_dep

Tools for displaying graphs of running ROS nodes with connecting topics and package dependancies respectively.



Usage:

```
$ rqt_graph
$ rqt_dep
```

Development Environments

rqt_shell, and rqt_py_console

Two tools for accessing an xterm shell and python console respectively.
Usage:

```
$ rqt
Plugin Menu->Miscellaneous Tools->Shell
Plugin Menu->Miscellaneous Tools->Python Console
```

Data Visualization Tools

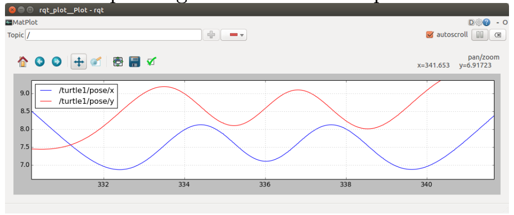
view_frames

A tool for visualizing the full tree of coordinate transforms.
Usage:

```
$ rosrun tf2_tools view_frames.py
$ evince frames.pdf
```

rqt_plot

A tool for plotting data from ROS topic fields.



Examples:
To graph the data in different plots:

```
$ rqt_plot /topic1/field1 /topic2/field2
```

To graph the data all on the same plot:

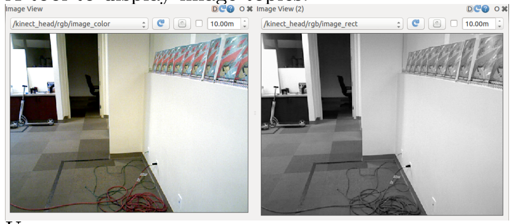
```
$ rqt_plot /topic1/field1,/topic2/field2
```

To graph multiple fields of a message:

```
$ rqt_plot /topic1/field1:field2:field3
```

rqt_image_view

A tool to display image topics.



Usage:

```
$ rqt_image_view
```

ROS Kinetic Catkin Workspaces

Create a catkin workspace

Setup and use a new catkin workspace from scratch.
Example:

```
$ source /opt/ros/kinetic/setup.bash
$ mkdir -p ~/catkin_ws/src
$ cd ~/catkin_ws/src
$ catkin_init_workspace
```

Checkout an existing ROS package

Get a local copy of the code for an existing package and keep it up to date using wstool.
Examples:

```
$ cd ~/catkin_ws/src
$ wstool init
$ wstool set tut --git git://github.com/ros/ros_tutorials.git
$ wstool update
```

Create a new catkin ROS package

Create a new ROS catkin package in an existing workspace with catkin create package.
Usage:

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

Example:

```
$ cd ~/catkin_ws/src
$ catkin_create_pkg tutorials std_msgs rospy roscpp
```

Build all packages in a workspace

Use catkin_make to build all the packages in the workspace and then source the setup.bash to add the workspace to the ROS_PACKAGE_PATH.
Examples:

```
$ cd ~/catkin_ws
$ catkin_make
$ source devel/setup.bash
```

CMakeLists.txt

Your CMakeLists.txt file MUST follow this format otherwise your packages will not build correctly.

`cmake_minimum_required()` Specify the name of the package
`project()` Project name which can refer as ${PROJECT_NAME}
`find_package()` Find other packages needed for build
`catkin_package()` Specify package build info export

Build Executables and Libraries:
Use CMake function to build executable and library targets. These macro should call after `catkin_package()` to use `catkin_*` variables.

```
include_directories(include ${catkin_INCLUDE_DIRS})
add_executable(hoge src/hoge.cpp)
add_library(fuga src/fuga.cpp)
target_link_libraries(hoge fuga ${catkin_LIBRARIES})
```

Message generation:
There are `add_{message,service,action}_files()` macros to handle messages,services and actions respectively. They must call before `catkin_package()`.

```
find_package(catkin COMPONENTS message_generation std_msgs)
add_message_files(FILES Message1.msg)
generate_messages(DEPENDENCIES std_msgs)
catkin_package(CATKIN_DEPENDS message_runtime)
```

If your package builds messages as well as executables that use them, you need to create an explicit dependency.

```
add_dependencies(hoge ${PROJECT_NAME}_generate_messages_cpp)
```

Open Source Robotics Foundation ROS