

# ROS Indigo Cheatsheet

## Filesystem Management Tools

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

### roslaunch

Runs a ROS package's executable with minimal typing.

Usage:

```
$ roslaunch package_name executable_name
```

Example (runs [turtlesim](#)):

```
$ roslaunch 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

### roscd

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

Commands:

<b>roscd ping</b>	Test connectivity to node.
<b>roscd list</b>	List active nodes.
<b>roscd info</b>	Print information about a node.
<b>roscd machine</b>	List nodes running on a machine.
<b>roscd kill</b>	Kill a running node.

Examples:

```
Kill all nodes:
$ roscd kill -a
List nodes on a machine:
$ roscd machine aqy.local
Ping all nodes:
$ roscd ping --all
```

### rostopic

A tool for displaying information about ROS [topics](#), including publishers, subscribers, publishing rate, and messages.

Commands:

<b>rostopic bw</b>	Display bandwidth used by topic.
<b>rostopic echo</b>	Print messages to screen.
<b>rostopic find</b>	Find topics by type.
<b>rostopic hz</b>	Display publishing rate of topic.
<b>rostopic info</b>	Print information about an active topic.
<b>rostopic list</b>	List all published topics.
<b>rostopic pub</b>	Publish data to topic.
<b>rostopic type</b>	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 rosmmsg to view the msg type:
$ rostopic type /topic_name | rosmmsg show
```

### rosservice

A tool for listing and querying ROS services.

Commands:

<b>rosservice list</b>	Print information about active services.
<b>rosservice node</b>	Print name of node providing a service.
<b>rosservice call</b>	Call the service with the given args.
<b>rosservice args</b>	List the arguments of a service.
<b>rosservice type</b>	Print the service type.
<b>rosservice uri</b>	Print the service ROSRPC uri.
<b>rosservice find</b>	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
```

### roscd

A tool for getting and setting ROS [parameters](#) on the parameter server using YAML-encoded files.

Commands:

<b>roscd set</b>	Set a parameter.
<b>roscd get</b>	Get a parameter.
<b>roscd load</b>	Load parameters from a file.
<b>roscd dump</b>	Dump parameters to a file.
<b>roscd delete</b>	Delete a parameter.
<b>roscd list</b>	List parameter names.

Examples:

```
List all the parameters in a namespace:
$ roscd list /namespace
Setting a list with one as a string, integer, and float:
$ roscd set /foo "[1', 1, 1.0]"
Dump only the parameters in a specific namespace to file:
$ roscd dump yml /namespace
```

### rosmmsg/rossrv

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

Commands:

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

Examples:

```
Display the Pose msg:
$ rosmmsg show Pose
List the messages in the nav_msgs package:
$ rosmmsg package nav_msgs
List the packages using sensor_msgs/CameraInfo:
$ rosmmsg packages sensor_msgs/CameraInfo
```

## Logging Tools

### roscd

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

Commands:

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

Examples:

```
Record select topics:
$ roscd record topic1 topic2
Replay all messages without waiting:
$ roscd play -a demo.log.bag
Replay several bag files at once:
$ roscd 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:

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

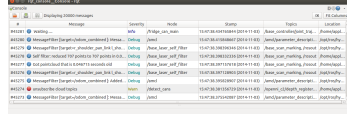
Examples:

```
To echo the transform between /map and /odom:
$ roscd tf tf_echo /map /odom
```

## Logging Tools

### rqt\_console

A tool to display and filtering messages published on rosout.

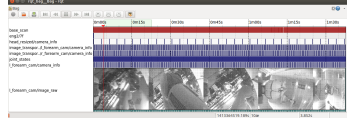


Usage:

```
$ 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

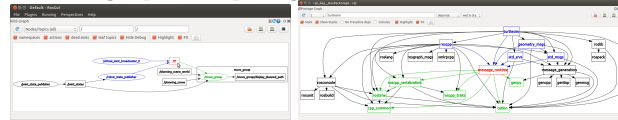
A tool for dynamically reconfiguring ROS parameters.

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 dependencies 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

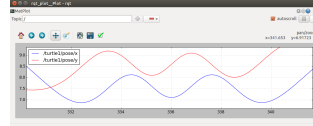
A tool for visualizing the full tree of coordinate transforms.

Usage:

```
$ rosrn 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:

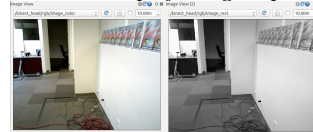
```
$ rqt_plot /topic1/field1,/topic2/field2
```

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 Indigo Catkin Workspaces

### Create a catkin workspace

Setup and use a new catkin workspace from scratch.

Example:

```
$ source /opt/ros/indigo/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 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)
```