ROS Cheat Sheet

Filesystem Command-line Tools

rospack/rosstack	A tool inspecting packages/stacks.
roscd	Changes directories to a package or stack.
rosls	Lists package or stack information.
roscreate-pkg	Creates a new ROS package.
roscreate-stack	Creates a new ROS stack.
rosdep	Installs ROS package system dependencies.
rosmake	Builds a ROS package.
roswtf	Displays a errors and warnings about a running ROS system or launch file.
rxdeps	Displays package structure and dependencies.

Usage:

```
$ rospack find [package]
$ roscd [package[/subdir]]
$ rosls [package[/subdir]]
$ roscreate-pkg [package_name]
$ rosmake [package]
$ rosdep install [package]
$ roswtf or roswtf [file]
$ rxdeps [options]
```

Common Command-line Tools

roscore

A collection of nodes and programs that are pre-requisites of a ROS-based system. You must have a roscore running in order for ROS nodes to communicate.

roscore is currently defined as:

```
master
parameter server
rosout
```

Usage:

```
$ roscore
```

rosmsg/rossrv

rosmsg/rossrv displays Message/Service (msg/srv) data structure definitions.

Commands:

rosmsg show	Display the fields in the msg.
rosmsg users	Search for code using the msg.
rosmsg md5	Display the msg md5 sum.
rosmsg package	List all the messages in a package.
rosnode packages	List all the packages with messages.

Examples:

Display the Pose msg:
```
$ rosmsg show Pose
```
List the messages in nav_msgs:
```
$ rosmsg package nav_msgs
```
List the files using sensor_msgs/CameraInfo:
```
$ rosmsg users sensor_msgs/CameraInfo
```

rosrun

rosrun allows you to run an executable in an arbitrary package without having to cd (or roscd) there first.

Usage:
```
$ rosrun package executable
```

Example:

Run turtlesim:
```
$ rosrun turtlesim turtlesim_node
```

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 particular machine.
rosnode kill	Kills a running node.

Examples:

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

roslaunch

Starts ROS nodes locally and remotely via SSH, as well as setting parameters on the parameter server.

Examples:

Launch on a different port:
```
$ roslaunch -p 1234 package filename.launch
```
Launch a file in a package:
```
$ roslaunch package filename.launch
```
Launch on the local nodes:
```
$ roslaunch --local package filename.launch
```

rostopic

A tool for displaying debug 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 hz	Display publishing rate of topic.
rostopic list	Print information about active topics.
rostopic pub	Publish data to topic.
rostopic type	Print topic type.
rostopic find	Find topics by 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
```

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
```

rosservice

A tool for listing and querying ROS services.

Commands:

rosservice list	Print information about active services.
rosservice node	Print the name of the 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
```

Logging Command-line Tools

rosbag

This is a set of tools for recording from and playing back to ROS topics. It is intended to be high performance and avoids deserialization and reserializationof the messages.

rosbag record will generate a ".bag" file (so named for historical reasons) with the contents of all topics that you pass to it.

Examples:
  Record all topics:
```
$ rosbag record -a
```
  Record select topics:
```
$ rosbag record topic1 topic2
```

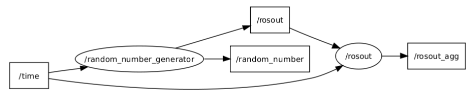
rosbag play will take the contents of one or more bag file, and play them back in a time-synchronized fashion.

Examples:
  Replay all messages without waiting:
```
$ rosbag play -a demo_log.bag
```
  Replay several bag files at once:
```
$ rosbag play demo1.bag demo2.bag
```

Graphical Tools

rxgraph

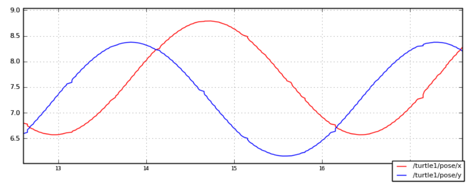
Displays a graph of the ROS nodes that are currently running, as well as the ROS topics that connect them.



Usage:
```
$ rxgraph
```

rxplot

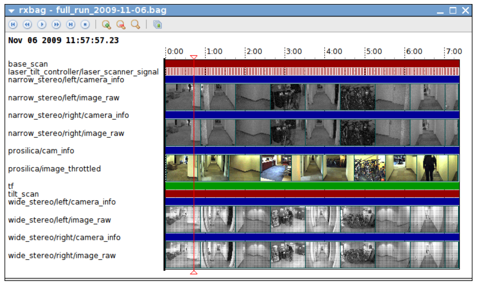
A tool for plotting data from one or more ROS topic fields using matplotlib.



Examples:
  To graph the data in different plots:
```
$ rxplot /topic1/field1 /topic2/field2
```
  To graph the data all on the same plot:
```
$ rxplot /topic1/field1,/topic2/field2
```
  To graph multiple fields of a message:
```
$ rxplot /topic1/field1:field2:field3
```

rxbag

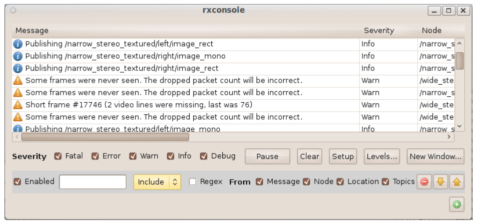
A tool for visualizing, inspecting, and replaying histories (bag files) of ROS messages.


rxbag - full_run_2009-11-06.bag
Nov 06 2009 11:57:57.23
0:00
1:00
2:00
3:00
4:00
5:00
6:00
7:00
base_scan
laser_tilt_controller/laser_scanner_signal
narrow_stereo/left/camera_info
narrow_stereo/left/image_raw
narrow_stereo/right/camera_info
narrow_stereo/right/image_raw
prosilica/cam_info
prosilica/image_throttled
tf
tilt_scan
wide_stereo/left/camera_info
wide_stereo/left/image_raw
wide_stereo/right/camera_info
wide_stereo/right/image_raw

Usage:
```
$ rxbag bag_file.bag
```

rxconsole

A tool for displaying and filtering messages published on rosout.



```
Message                                                               Severity  Node
Publishing /narrow_stereo_textured/left/image_rect                    Info      /narrow_s
Publishing /narrow_stereo_textured/right/image_mono                   Info      /narrow_s
Publishing /narrow_stereo_textured/right/image_rect                   Info      /narrow_s
Some frames were never seen. The dropped packet count will be incorrect.  Warn  /wide_ste
Some frames were never seen. The dropped packet count will be incorrect.  Warn  /narrow_s
Short frame #17746 (2 video lines were missing, last was 76)          Warn      /wide_ste
Some frames were never seen. The dropped packet count will be incorrect.  Warn  /wide_ste
Publishing /narrow_stereo_textured/left/image_mono                    Info      /narrow_s
```

Usage:
```
$ rxconsole
```

tf Command-line Tools

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
```

view_frames

A tool for visualizing the full tree of coordinate transforms.

Usage:
```
$ rosrun tf view_frames
$ evince frames.pdf
```