# ROS 2 Command Line Interface Reference

This document provides a reference for selected ROS 2 CLI subcommands, with color-coded commands and short descriptions for their subcommands as provided in the input. Only the subcommands listed in the input file are included, with additional sections for launch files, TurtleBot3, and translating .xml launch files.

## colcon

A build tool for ROS 2 workspaces.

- `colcon build` : Builds all packages in the workspace.
- `colcon build --packages-select <package_name>` : Builds only the specified package.
- `colcon build --packages-select <package_name> --symlink-install` : Builds the specified package with symbolic links for faster iteration.

## rqt

A Qt-based GUI framework for ROS 2.

- `rqt_graph` : Displays a graphical representation of the ROS 2 node and topic network.

## rviz2

RViz2 is a 3D visualization tool for ROS 2. It allows users to visualize and interact with robot data, sensor information, transform frames, and other relevant data in a 3D environment.

- `ros2 run rviz2 rviz2` : Opens empty rviz2 environment.
- `ros2 run rviz2 rviz2 -d <rviz_config_path>` : Opens rviz2 environment with a config file saved from a previous session.

## ros2 run

Executes a ROS 2 node from a package.

- `ros2 run -h` : Displays help information for the run command.
- `ros2 run <package_name> <node_executable>` : Runs the specified node from the given package.
- `ros2 run <package_name> <node_executable> --ros-args -r __node:=<new_node_name>` : Runs the node with a remapped node name.
- `ros2 run <package_name> <node_executable> --ros-args -r <topic_name>:=<new_topic_name>` : Runs the node with a remapped topic name.

- `ros2 run <package_name> <node_executable> --ros-args -r <service_name>:=<new_service_name>` : Runs the node with a remapped service name.
- `ros2 run <package_name> <node_executable> --ros-args -p <param_name>:=<custom_param_value>` : Runs the node with a remapped parameter.
- `ros2 run tf2_tools view_frames -o <graph_name>` : How to generate a link-joint graph, as seen in "my_robot_frames.pdf"

## ros2 node

Manages ROS 2 nodes.

- `ros2 node -h` : Displays help information for the node command.
- `ros2 node list` : Lists all active ROS 2 nodes.
- `ros2 node info </node_name>` : Provides detailed information about a specific node.

## ros2 interface

Inspects ROS 2 interfaces (topics, services, actions).

- `ros2 interface -h` : Displays help information for the interface command.
- `ros2 interface list` : Lists all available ROS 2 interfaces.
- `ros2 interface show <interface_type>` : Displays the definition of a specific interface.
- `ros2 interface package <package_name>` : Lists interfaces defined in the specified package.

## ros2 topic

Interacts with ROS 2 topics.

- `ros2 topic -h` : Displays help information for the topic command.
- `ros2 topic list` : Lists all active ROS 2 topics.
- `ros2 topic echo </topic_name>` : Prints data published to a specific topic.
- `ros2 topic info </topic_name>` : Provides detailed information about a specific topic.
- `ros2 topic hz </topic_name>` : Measures the publishing rate of a topic.
- `ros2 topic bw </topic_name>` : Measures the bandwidth usage of a topic.
- `ros2 topic pub -r <rate> </topic_name> <message_type> <data>` : Publishes data to a topic at a specified rate.

## ros2 service

Interacts with ROS 2 services.

- `ros2 service -h` : Displays help information for the service command.
- `ros2 service list` : Lists all active ROS 2 services.
- `ros2 service info </service_name>` : Provides detailed information about a specific service.
- `ros2 service type </service_name>` : Displays the interface type of a specific service.
- `ros2 service call </service_name> <service_type> <data>` : Calls a service with specified data.

## ros2 launch

Launches a ROS 2 launch file to start multiple nodes and configurations.

- `ros2 launch -h` : Displays help information for the launch command.
- `ros2 launch <package_name> <launch_file>` : Launches the specified launch file from the package.
- `ros2 launch <package_name> <launch_file> <arg_name>:=<value>` : Launches the launch file with custom argument values.
- `ros2 launch <package_name> <launch_file> --show-args` : Shows available arguments for the launch file.
- `ros2 launch <package_name> <launch_file> -d` : Launches the launch file in debug mode.
- `ros2 launch <package_name> <launch_file> -p` : Prints the launch description without executing.

## TurtleBot3

Common commands for launching and controlling TurtleBot3 in Gazebo simulations, including SLAM and navigation with RViz.

- `export TURTLEBOT3_MODEL=burger` : Sets the TurtleBot3 model to burger.
- `export TURTLEBOT3_MODEL=waffle` : Sets the TurtleBot3 model to waffle.
- `export TURTLEBOT3_MODEL=waffle_pi` : Sets the TurtleBot3 model to waffle_pi.
- `ros2 launch turtlebot3_gazebo empty_world.launch.py` : Launches TurtleBot3 in an empty Gazebo world.

- `ros2 launch turtlebot3_gazebo turtlebot3_world.launch.py` : Launches TurtleBot3 in the TurtleBot3 world.
- `ros2 launch turtlebot3_gazebo turtlebot3_house.launch.py` : Launches TurtleBot3 in the TurtleBot3 house.
- `ros2 run turtlebot3_teleop teleop_keyboard` : Runs the keyboard teleoperation node for TurtleBot3.
- `ros2 launch turtlebot3_cartographer cartographer.launch.py use_sim_time:=true` : Launches Cartographer SLAM with RViz for mapping in simulation (run after launching Gazebo).
- `ros2 run nav2_map_server map_saver_cli -f ~/map` : Saves the generated map after SLAM.
- `ros2 launch turtlebot3_navigation2 navigation2.launch.py use_sim_time:=true map:=$HOME/map.yaml` : Launches Navigation2 with RViz for navigation in simulation (run after launching Gazebo and with a saved map).

## Common Commands for Custom Robot

Common commands for launching and controlling the custom robot from this repository in Gazebo simulations, including SLAM and navigation with RViz2. Parse the launch file line-by-line to further understand how the world and robot are launched.

- `ros2 launch custom_robot_bringup custom_robot_empty_gz.launch.xml` : Launches custom robot in an empty world.
- `ros2 launch custom_robot_bringup custom_robot_maze_gz.launch.xml` : Launches custom robot in the maze world.
- `ros2 run teleop_twist_keyboard teleop_twist_keyboard` : Runs the keyboard teleoperation node for the custom robot.
- `ros2 run robot_state_publisher robot_state_publisher --ros-args -p robot_description:="$(xacro $urdf_path)"` : Provides the robot URDF as input to the robot state publisher node. This node takes as input the robot URDF and the published joint states to output transforms used by ros2 packages (e.g., Nav2) downstream.

## Translating .xml launch files

- `<let>` declares variables you can use later in .xml files
- `<node>` mimics `ros2 run <package_name> <node_executable>`
  - `<param name="" value="" />` mimics `--ros-args -p <param_name>:=<custom_param_value>`
- `<include>` allows you to include another launch file in this launch file.

- `<include file=""> <arg name="" value="" /> </include>` mimics `ros2 launch <package_name> <launch_file> <arg_name>:=<value>`

For example:

```
<launch>
    <let name="urdf_path"
         value="$(find-pkg-share custom_robot_description)/urdf/custom_robot.urdf
         .xacro" />

    <node pkg="robot_state_publisher" exec="robot_state_publisher">
        <param name="robot_description"
               value="$(command 'xacro $(var urdf_path)')" />
    </node>

    <include file="$(find-pkg-share ros_gz_sim)/launch/gz_sim.launch.py">
        <arg name="gz_args" value="$(find-pkg-share custom_robot_bringup)/worlds/
        maze_world.sdf -r" />
    </include>

</launch>
```

is equivalent to executing the following commands in a terminal:

```
$ export urdf_path="$(ros2 pkg prefix --share custom_robot_description)/urdf/
custom_robot.urdf.xacro"
$ ros2 run robot_state_publisher robot_state_publisher --ros-args -p
robot_description:="$(xacro $urdf_path)"
$ ros2 launch ros_gz_sim gz_sim.launch.py gz_args:="$(ros2 pkg prefix --share
custom_robot_bringup)/worlds/maze_world.sdf -r"
```