ROS 2 Commands Cheat Sheet

General Format of ROS 2 CLI

The keyword '**ros2**' is the unique entry point for the CLI.
Every ROS 2 command starts with the **ros2** keyword, followed by a command, a verb, and possibly positional/optional arguments.

ros2 [command] [verb] <positional-argument> <optional-arguments>

For Help on ROS 2 CLI commands-

$ ros2 [command] --help
$ ros2 [command] [verb] -h

Action

A type of message-based communication that allows a client node to request a specific goal to be achieved by a server node, and receive feedback and/or a result from the server node once the goal has been completed.

Command

ros2 action [verb] <arguments>

Verbs

list: identify all the actions in the ROS graph
info <action_name>: introspect about an action
send_goal <action_name> <action_type> <values>: send an action goal

Arguments

-f: echo feedback messages for the goal

Examples

$ ros2 action list
$ ros2 action info /turtle1/rotate_absolute
$ ros2 action send_goal /turtle1/rotate_absolute turtlesim/action/RotateAbsolute "{theta: 1.57}"
$ ros2 interface show turtlesim/action/RotateAbsolute

Bag

A file format used to record and playback ROS 2 topics.

Command

ros2 bag [verb] <arguments>

Verbs

record <topic_name>: record the data published to topic
info <bag_file_name>: get details about bag file
play <bag_file_name>: replaying the bag file

Arguments

--clock: publish to /clock at a specific frequency in Hz
-l: enable loop playback when playing a bag file
-r: rate to play back messages
-s: storage identifier to be used, defaults to 'sqlite3'
--topics: topics to replay, separated by space
--storage-config-file: path to a yaml file defining storage specific configurations
-a: recording all topics, required if no topics are listed explicitly or through a regex
-e: recording only topics matching provided regular expression
-x: exclude topics matching provided regular expression

ROS 2 Commands Cheat Sheet

-o: destination of the bag file to create

Examples

$ ros2 bag record /turtle1/cmd_vel
$ ros2 bag record -o my_bag /turtle1/cmd_vel /turtle1/pose
$ ros2 bag info <bag_name.db3>
$ ros2 bag play <bag_name.db3>

Component

A modular unit of software that encapsulates functionality, data, and communication.

****ROS 2 Components ≈ ROS 1 Nodelets***

Command

ros2 component [verb] <arguments>

Verbs

list: output a list of running containers and components
load: load a component into a container node
standalone: run a component into its own standalone container node
types: output a list of components registered in the ament index
unload: unload a component from a container node

Arguments

-n: component node name
--node-namespace: component node namespace
--log-level: component node log level
-r: component node remapping rules, in the 'from:=to' form
-p: component node parameters, in the 'name:=value' form

Examples

$ ros2 component list
$ ros2 component types
$ ros2 component load /ComponentManager composition composition::Talker

Control

A control framework to simplify integrating new hardware and overcome some drawbacks.

Command

ros2 control [verb] <arguments>

Verbs

list_controller_types: output the available controller types and their base classes
list_controllers: output the list of loaded controllers, their type, and status
list_hardware_interfaces: output the list of loaded controllers, their type and status
load_controller: load a controller in a controller manager
reload_controller_libraries: reload controller libraries
set_controller_state: adjust the state of the controller
switch_controllers: switch controllers in a controller manager
unload_controller: unload a controller in a controller manager
view_controller_chains: generates a diagram of the loaded chained controllers

Arguments

-c: name of the controller manager ROS node

ROS 2 Commands Cheat Sheet

--claimed-interfaces: list controller's claimed interfaces
--required-state-interfaces: list controller's required state interfaces
--required-command-interfaces: list controller's required command interfaces

Examples

$ ros2 control list_controllers
$ ros2 control list_hardware_components -h
$ ros2 control list_hardware_interfaces

Daemon

A system-level process that runs in the background and provides various services to ROS 2 nodes and components.

***ROS 2 Daemon ≈ ROS 1 Master*

Command

ros2 daemon [verb]

Verbs

start: start the daemon if it isn't running
status: output the status of the daemon
stop: stop the daemon if it is running

Examples

$ ros2 daemon start
$ ros2 daemon status
$ ros2 daemon stop

Doctor

Checks all aspects of ROS 2, and warns about possible errors and reasons for issues.

Command

ros2 doctor <arguments>

Arguments

--report: print all warnings
-rf: print reports of failed checks only
-iw: include warnings as failed checks. Warnings are ignored by default

Examples

$ ros2 doctor
$ ros2 doctor --report
$ ros2 doctor --include-warnings

Extension Point

Lists extension points.

Command

ros2 extension_points <arguments>

Arguments

--all, -a: show extension points which failed to be imported
--verbose, -v: show more information for each extension point

Examples

$ ros2 extension_points
$ ros2 extension_points --all

Extension

Lists extensions of a package.

Command

ros2 extensions <arguments>

Arguments

-a: show extensions which failed to load or are incompatible
-v: show more information for each extension

Examples

$ ros2 extensions
$ ros2 extensions --all

ROS 2 Commands Cheat Sheet

Interface

ROS applications typically communicate through interfaces of one of three types: messages, services, and actions.

Command

ros2 interface [verb]

Verbs

list: list all interface types available

package <package_name>: output a list of available interface types within one package

packages: output a list of packages that provide interfaces

show <interface_type>: output the interface definition

Examples

$ ros2 interface list

$ ros2 interface package turtlesim

$ ros2 interface show turtlesim/msg/Pose

$ ros2 interface packages

Launch File

Allows to execute multiple nodes with their own complete configuration (remapping, parameters, etc.) in a single file, that can launch with only one command line.

Command

ros2 launch [package_name] [launch_file_name] <arguments>

Argument

-n, --noninteractive: run the launch system non-interactively, with no terminal associated

-d, --debug: put the launch system in debug mode, provides more verbose output.

-p, --print, --print-description: print the launch description to the console without launching it.

-s, --show-args, --show-arguments: show arguments that may be given to the launch file.

--show-all-subprocesses-output, -a: show all launched subprocesses output by overriding their output configuration using the OVERRIDE_LAUNCH_PROCESS_OUTPUT envvar.

--launch-prefix LAUNCH_PREFIX: prefix command, which should go before all executables. It must be wrapped in quotes if it contains spaces (e.g. --launch-prefix 'xterm -e gdb -ex run --args').

--launch-prefix-filter LAUNCH_PREFIX_FILTER: regex pattern for filtering which executables the --launch-prefix is applied to by matching the executable name.

Examples

$ ros2 launch my_first_launch_file.launch.py

$ ros2 launch my_first_launch_file.launch.py --noninteractive

$ ros2 launch my_first_launch_file.launch.py --show-all-subprocesses-output

Lifecycle

Manages node containing a state machine with a set of predefined states. These states can be changed by invoking a transition id which indicates the succeeding consecutive state.

Command

ros2 lifecycle [verb]

ROS 2 Commands Cheat Sheet

Verbs

list <node_name>: output a list of available transitions

get: get lifecycle state for one or more nodes

nodes: output a list of nodes with lifecycle

set: trigger lifecycle state transition

Examples

$ ros2 lifecycle list

$ ros2 lifecycle get

Multicast

In order to communicate successfully via DDS, the used network interface has to be multicast enabled.

Command

ros2 multicast [verb]

Verbs

receive: receive a single UDP multicast packet

send: send a single UDP multicast packet

Examples

$ ros2 multicast receive

$ ros2 multicast send

Node

An executable within a ROS 2 package that performs computation and uses client libraries to communicate with other nodes

Command

ros2 run [package_name] [executable_name] <arguments>

Arguments

--prefix PREFIX: prefix command, before the executable. Must be wrapped in quotes if it contains spaces

–ros-args: pass arguments while executing a node

–remap: rename topics name while executing node

Examples

$ ros2 run turtlesim turtlesim_node

$ ros2 run turtlesim turtlesim_node --ros-args --remap __node:=my_turtle

Command

ros2 node [command]

Verbs

list: gives the names of all running nodes

info <node_name>: access more information about a node

Examples

$ ros2 node info /my_turtle

Package

An organisational unit for ROS 2 code and promote software reuse.

Create ROS2 package

$ cd <workspace_name>/src

Python package

$ ros2 pkg create –build-type ament_python [package_name]

C++ package

$ ros2 pkg create –build-type ament_cmake [package_name]

Parameter

A list of configuration values attached to a node.

Command

ros2 param [verb] <arguments>

ROS 2 Commands Cheat Sheet

Verbs

param list: see the parameters belonging to nodes

param get <node_name> <parameter_name>: display the type and current value of a parameter

param set <node_name> <parameter_name> <value>: change a parameter's value at runtime

ros2 param dump <node_name>: view a node's current parameter values

ros2 param load <node_name> <parameter_file>: load parameters from a file to a currently running node

Arguments

--output-dir: the absolute path to save the generated parameters file

Examples

```
$ ros2 param list
$ ros2 param get /turtlesim background_g
$ ros2 param set /turtlesim background_r 150
$ ros2 param dump /turtlesim > turtlesim.yaml
$ ros2 param load /turtlesim turtlesim.yaml
```

rqt tools

The rqt tools allow graphical representations of ROS nodes, topics, messages, and other information.

Command

rqt_graph: view the nodes and topics that are active

rqt: brings up a display screen, drop-down menu items to visualize various sources of data

Security

The sros2 package provides the tools and instructions to use ROS 2 on top of DDS-Security.

Command

ros2 security [verb]

Verbs

create_enclave: create enclave

create_keystore: create keystore

create_permission: create permission

generate_artifacts: generate keys and permission files from a list of identities and policy files

generate_policy: generate XML policy file from ROS graph data

list_enclaves: list enclaves in keystore

Examples

```
$ ros2 security create_keystore demo_keystore
$ ros2 security create_enclave demo_keystore /talker_listener/talker
$ ros2 security create_enclave demo_keystore /talker_listener/listener
```

Service

Communication-based on a call-and-response model, services only provide data when they are specifically called by a client.

Command

ros2 service [verb] <arguments>

ROS 2 Commands Cheat Sheet

Verbs

list: return a list of all the services currently active in the system

type <service_name>: find out the type of a service

find <type_name>: to find all the services of a specific type

call <service_name> <service_type> <arguments>: call a service

Arguments

-r: repeat the call at a specific rate in Hz

Examples

$ ros2 service list

$ ros2 service find std_srvs/srv/Empty

$ ros2 service call /spawn turtlesim/srv/Spawn "{x: 2, y: 2, theta: 0.2, name: ''}"

$ ros2 interface show turtlesim/srv/Spawn

Topic

Element of the ROS graph that acts as a bus for nodes to exchange messages.

Command

ros2 topic [verb] <arguments>

Verbs

list: return a list of all the topics

info <topic_name>: access more information about topics

echo <topic_name>: see the data being published on a topic

pub <--once> <topic_name> <msg_type> '<args>': publish data onto a topic directly from the command line

Arguments

-r: publishing rate in Hz (default: 1)

-p: only print every N-th published message

-1, --once: publish one message and exit

-t: publish this number of times and exit

--keep-alive: keep publishing node alive for N seconds after the last msg

Examples

$ ros2 topic info /turtle1/cmd_vel

$ ros2 topic echo /turtle1/cmd_vel

$ ros2 topic pub --once /turtle1/cmd_vel geometry_msgs/msg/Twist "{linear: {x: 2.0, y: 0.0, z: 0.0}, angular: {x: 0.0, y: 0.0, z: 1.8}}"

$ ros2 topic hz /turtle1/pose

Workspace

Directory containing ROS 2 packages.

Create a ROS 2 workspace directory

$ mkdir -p <workspace_name>/src

Build & Source workspace

$ cd <workspace_name>

$ colcon build

$ source install/setup.bash

ROS 2 Commands Cheat Sheet

Colcon Tools

colcon is a command line tool to improve the workflow of building, testing and using multiple software packages. Every **colcon** command starts with the **colcon** keyword, followed by a verb and possible arguments.

colcon [verb] <argument>

For Help on colcon CLI commands-

$ colcon --help

$ colcon [verb] -h

colcon build

Build a set of packages.

Command

colcon build <arguments>

Arguments

--build-base BUILD_BASE: base path for all build directories

--install-base INSTALL_BASE: base path for all install prefixes

--merge-install: install prefix for all packages instead of a package-specific subdirectory in the install base

--symlink-install: use symlinks instead of copying files from the source

--test-result-base TEST_RESULT_BASE: base path for all test results

--continue-on-error: continue building other packages when a package fails to build

--executor sequential: process one package at a time

–executor parallel: process multiple jobs in parallel

--packages-select PKG_NAME: select the packages with the passed names

--packages-skip PKG_NAME: skip the packages with the passed names

--parallel-workers NUMBER: maximum number of jobs to process in parallel. The default value is the number of logical CPU cores

Examples

$ colcon build

$ colcon build –build-base build

$ colcon build –install-base install

$ colcon build –merge-install

$ colcon build --symlink-install

$ colcon build --executor sequential

$ colcon build --packages-select my_pkg

$ colcon build --parallel-workers 5

colcon graph

Generates a visual representation of the package dependency graph.

Command

colcon graph <arguments>

Arguments

–density: output the density of the dependency graph

--legend: output a legend for the dependency graph

--dot: output topological graph in DOT (graph description language)

--dot-cluster: cluster packages by their file system path

Examples

$ colcon graph --density

$ colcon graph --legend

$ colcon graph --dot

$ colcon graph --dot-cluster

ROS 2 Commands Cheat Sheet

colcon info

Shows detailed information about packages.

Command

colcon info <arguments>

Arguments

PKG_NAME: explicit package names to only show their information

--base-paths: the base paths to recursively crawl for packages

--paths: the paths to check for a package

--packages-select: only process a subset of packages

--packages-skip: skip a set of packages

Examples

$ colcon info

$ colcon info --paths ros2_ws/src/*

$ colcon info --packages-select my_pkg

$ $ colcon info --base-paths pkg_dir_name

colcon list

Enumerates a set of packages.

Command

colcon list <arguments>

Arguments

--topological-order, -t: order output based on topological ordering

--names-only, -n: output only the name of each package but not the path and type

--paths-only, -p: output only the path of each package but not the name and type

Examples

$ colcon list --topological-order

$ colcon list --names-only

$ colcon list --paths-only

colcon test

Runs the tests for a set of packages.

Command

colcon test <arguments>

Arguments

--build-base BUILD_BASE: base path for all build directories

--install-base INSTALL_BASE: base path for all install prefixes

--merge-install: install prefix for all packages instead of a package-specific subdirectory in the install base

--test-result-base TEST_RESULT_BASE: base path for all test results

--retest-until-fail N: rerun tests up to N times if they pass.

--retest-until-pass N: rerun failing tests up to N times.

--abort-on-error: abort after the first package with any errors.

--return-code-on-test-failure: use a non-zero return code to indicate any test failure.

Examples

$ colcon test --test-result-base ./build-test

$ colcon test --retest-until-fail 5

$ colcon test --abort-on-error

$ colcon test --return-code-on-test-failure

ROS 2 Commands Cheat Sheet

colcon test-result

Summarises the results of previously run tests.

Command

colcon test-result <arguments>

Arguments

--test-result-base
TEST_RESULT_BASE: base path for all test results
--all: show all test result files including the ones without errors/failures.
--verbose: show additional information for each error/failure.
--result-files-only: print only the paths of the result files.
--delete: delete all result files.
--delete-yes: same as –delete without an interactive confirmation.

Examples

$ colcon test-result --test-result-base ./build-test
$ colcon test-result -all
$ colcon test-result --result-files-only