

ROS2 Command Cheat Sheet

P. 1

ROS 2 Basics

ROS 2 Cheats Sheet

Command Line Interface

All ROS 2 CLI tools start with the prefix 'ros2' followed by a command, a verb and (possibly) positional/optional arguments.

For any tool, the documentation is accessible with,

```
$ ros2 command --help
```

and similarly for verb documentation,

```
$ ros2 command verb -h
```

Similarly, auto-completion is available for all commands/verbs and most positional/optional arguments. E.g.,

```
$ ros2 command [tab][tab]
```

Some of the examples below rely on:

[ROS 2 demos package](#).

action Allows to manually send a goal and displays debugging information about actions.

Verbs:

```
info      Output information about an action.
list     Output a list of action names.
send_goal Send an action goal.
show    Output the action definition.
```

Examples:

```
$ ros2 action info /fibonacci
$ ros2 action list
$ ros2 action send_goal /fibonacci \
  action_tutorials/action/Fibonacci "order: 5"
$ ros2 action show action_tutorials/action/Fibonacci
```

bag Allows to record/play topics to/from a rosbag.

Verbs:

```
info     Output information of a bag.
play    Play a bag.
record  Record a bag.
```

Examples:

```
$ ros2 info <bag-name>
$ ros2 play <bag-name>
$ ros2 record -a
```

component Various component related verbs.

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.
```

Examples:

```
$ ros2 component list
$ ros2 component load /ComponentManager \
  composition composition::Talker
$ ros2 component types
$ ros2 component unload /ComponentManager 1
```

daemon Various daemon related verbs.

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
```

doctor A tool to check ROS setup and other potential issues such as network, package versions, rmw middleware etc.

Alias: **wtf** (where's the fire).

Arguments:

```
--report/-r      Output report of all checks.
--report-fail/-rf Output report of failed checks only.
--include-warning/-iw Include warnings as failed checks.
```

Examples:

```
$ ros2 doctor
$ ros2 doctor --report
$ ros2 doctor --report-fail
$ ros2 doctor --include-warning
$ ros2 doctor --include-warning --report-fail
```

or similarly,

```
$ ros2 wtf
```

extension_points List extension points.

extensions List extensions.

interface Various ROS interfaces (actions/topics/services)-related verbs. Interface type can be filtered with either of the following option, '--only-actions', '--only-msgs', '--only-srvs'.

Verbs:

```
list     List all interface types available.
package  Output a list of available interface types within one package.
packages Output a list of packages that provide interfaces.
proto    Print the prototype (body) of an interfaces.
show    Output the interface definition.
```

Examples:

```
$ ros2 interface list
$ ros2 interface package std_msgs
$ ros2 interface packages --only-msgs
$ ros2 interface proto example_interfaces/srv/AddTwoInts
$ ros2 interface show geometry_msgs/msg/Pose
```

launch Allows to run a launch file in an arbitrary package without to 'cd' there first.

Usage:

```
$ ros2 launch <package> <launch-file>
```

Example:

```
$ ros2 launch demo_nodes_cpp add_two_ints.launch.py
```

lifecycle Various lifecycle related verbs.

Verbs:

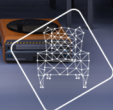
```
get     Get lifecycle state for one or more nodes.
list    Output a list of available transitions.
nodes  Output a list of nodes with lifecycle.
set    Trigger lifecycle state transition.
```

msg (**deprecated**) Displays debugging information about messages.

Verbs:

```
list     Output a list of message types.
package  Output a list of message types within a given package.
packages Output a list of packages which contain messages.
show    Output the message definition.
```

Examples:



ROS2 Command Cheat Sheet

p. 2

ROS 2 Basics

```
$ ros2 msg list
$ ros2 msg package std_msgs
$ ros2 msg packages
$ ros2 msg show geometry_msgs/msg/Pose
```

multicast Various multicast related verbs.

Verbs:

- receive** Receive a single UDP multicast packet.
- send** Send a single UDP multicast packet.

node Displays debugging information about nodes.

Verbs:

- info** Output information about a node.
- list** Output a list of available nodes.

Examples:

```
$ ros2 node info /talker
$ ros2 node list
```

param Allows to manipulate parameters.

Verbs:

- delete** Delete parameter.
- describe** Show descriptive information about declared parameters.
- dump** Dump the parameters of a given node in yaml format, either in terminal or in a file.
- get** Get parameter.
- list** Output a list of available parameters.
- set** Set parameter

Examples:

```
$ ros2 param delete /talker /use_sim_time
$ ros2 param get /talker /use_sim_time
$ ros2 param list
$ ros2 param set /talker /use_sim_time false
```

pkg Create a ros2 package or output package(s)-related information.

Verbs:

- create** Create a new ROS2 package.
- executables** Output a list of package specific executables.
- list** Output a list of available packages.
- prefix** Output the prefix path of a package.
- xml** Output the information contained in the package xml manifest.

Examples:

```
$ ros2 pkg executables demo_nodes_cpp
$ ros2 pkg list
$ ros2 pkg prefix std_msgs
$ ros2 pkg xml -t version
```

run Allows to run an executable in an arbitrary package without having to 'cd' there first.

Usage:

```
$ ros2 run <package> <executable>
```

Example:

```
$ ros2 run demo_node_cpp talker
```

security Various security related verbs.

Verbs:

- create_key** Create key.
- create_permission** Create keystore.
- generate_artifacts** Create permission.
- list_keys** Distribute key.
- create_keystore** Generate keys and permission files from a list of identities and policy files.
- distribute_key** Generate XML policy file from ROS graph data.
- generate_policy** List keys.

Examples (see [sros2 package](#)):

```
$ ros2 security create_key demo_keys /talker
$ ros2 security create_permission demo_keys /talker \
  policies/sample_policy.xml
$ ros2 security generate_artifacts
$ ros2 security create_keystore demo_keys
```

service Allows to manually call a service and displays debugging information about services.

Verbs:

- call** Call a service.
- find** Output a list of services of a given type.
- list** Output a list of service names.
- type** Output service's type.

Examples:

```
$ ros2 service call /add_two_ints \
  example_interfaces/AddTwoInts "a: 1, b: 2"
$ ros2 service find rcl_interfaces/srv/ListParameters
$ ros2 service list
$ ros2 service type /talker/describe_parameters
```

srv (deprecated) Various srv related verbs.

Verbs:

- list** Output a list of available service types.
- package** Output a list of available service types within one package.
- packages** Output a list of packages which contain services.
- show** Output the service definition.

test Run a ROS2 launch test.

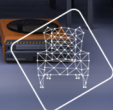
topic A tool for displaying debug information about ROS topics, including publishers, subscribers, publishing rate, and messages.

Verbs:

- bw** Display bandwidth used by topic.
- delay** Display delay of topic from timestamp in header.
- echo** Output messages of a given topic to screen.
- find** Find topics of a given type type.
- hz** Display publishing rate of topic.
- info** Output information about a given topic.
- list** Output list of active topics.
- pub** Publish data to a topic.
- type** Output topic's type.

Examples:

```
$ ros2 topic bw /chatter
$ ros2 topic echo /chatter
$ ros2 topic find rcl_interfaces/msg/Log
$ ros2 topic hz /chatter
$ ros2 topic info /chatter
$ ros2 topic list
$ ros2 topic pub /chatter std_msgs/msg/String \
  'data: Hello ROS 2 world'
$ ros2 topic type /rosout
```



ROS 2 Command Cheat Sheet

ROS 2 Compile

ROS 2 Cheats Sheet

colcon - collective construction

colcon is a command line tool to improve the workflow of building, testing and using multiple software packages. It automates the process, handles the ordering and sets up the environment to use the packages.

All colcon tools start with the prefix 'colcon' followed by a command and (likely) positional/optional arguments.

For any tool, the documentation is accessible with,

```
$ colcon command --help
```

Moreover, colcon offers auto-completion for all verbs and most positional/optional arguments. E.g.,

```
$ colcon command [tab][tab]
```

Find out how to enable auto-completion at [colcon's online documentation](#).

Environment variables:

- o CMAKE_COMMAND The full path to the CMake executable.
- o COLCON_ALL_SHELLS Flag to enable all shell extensions.
- o COLCON_COMPLETION_LOGFILE Set the logfile for completion time.
- o COLCON_DEFAULTS_FILE Set path to the yaml file containing the default values for the command line arguments (default:\$COLCON_HOME/defaults.yaml).
- o COLCON_DEFAULT_EXECUTOR Select the default executor extension.
- o COLCON_EXTENSION_BLACKLIST Blacklist extensions which should not be used.
- o COLCON_HOME Set the configuration directory (default: ~/.colcon.)
- o COLCON_LOG_LEVEL Set the log level (debug—10, info—20, warn—30, error—40, critical—50, or any other positive numeric value).
- o COLCON_LOG_PATH Set the log directory (default: \$COLCON_HOME/log).
- o CTEST_COMMAND The full path to the CTest executable.
- o POWERSHELL_COMMAND The full path to the PowerShell executable.

Global options:

- o --log-base <path> The base path for all log directories (default: log).
- o --log-level <level> Set log level for the console output, either by numeric or string value (default: warn)

build Build a set of packages.

Examples:

Build the whole workspace:

```
$ colcon build
```

Build a single package excluding dependencies:

```
$ colcon build --packages-selected demo_nodes_cpp
```

Build two packages including dependencies, use symlinks instead of copying files where possible and print immediately on terminal:

```
$ colcon build --packages-up-to demo_nodes_cpp \
  action_tutorials --symlink-install \
  --event-handlers console_direct+
```

extension-points List extension points.

extensions Package information.

info List extension points.

list List packages, optionally in topological ordering.

Example:

List all packages in the workspace:

```
$ colcon list
```

List all packages names in topological order up-to a given package:

```
$ colcon list --names-only --topological-order \
  --packages-up-to demo_nodes_cpp
```

metadata Manage metadata of packages.

test Test a set of packages.

Example:

Test the whole workspace:

```
$ colcon test
```

Test a single package excluding dependencies:

```
$ colcon test --packages-select demo_nodes_cpp
```

Test a package including packages that depend on it:

```
$ colcon test --packages-above demo_nodes_py
```

Test two packages including dependencies, and print on terminal:

```
$ colcon test --packages-up-to demo_nodes_cpp \
  demo_nodes_py --event-handlers console_direct+
```

Pass arguments to pytest (e.g. to print a coverage report):

```
$ colcon test --packages-select demo_nodes_cpp \
  --event-handlers console_direct+ \
  --pytest-args --cov=sros2
```

test-result Show the test results generated when testing a set of packages.

Example:

Show all test results generated, including successful tests:

```
$ colcon test-result --all
```

version-check Compare local package versions with PyPI.

Examples:

```
$ todo
```

Must know colcon flags.

- o --symlink-install Use 'symlinks' instead of installing (copying) files where possible.
- o --continue-on-error Continue other packages when a package fails to build. Packages recursively depending on the failed package are skipped.
- o --event-handlers console_direct+ Show output on console.
- o --event-handlers console_cohesion+ Show output on console after a package has finished.
- o --packages-select Build only specific package(s).
- o --packages-up-to Build specific package(s) and its/their recursive dependencies.
- o --packages-above Build specific package(s) and other packages that recursively depending on it.
- o --packages-skip Skip package(s).
- o --packages-skip-build-finished Skip a set of packages which have finished to build previously.
- o --cmake-args Pass arguments to CMake projects.
- o --cmake-clean-cache Remove CMake cache before the build (implicitly forcing CMake configure step).
- o --cmake-clean-first Build target 'clean' first, then build (to only clean use '-cmake-target clean').
- o --cmake-force-configure Force CMake configure step.

