ROS 2 Commands Cheat Sheet

Yassine Cherni

General Format of ROS 2 CLI

- **The keyword 'ros2':** This is the unique entry point for the CLI.
- Every ROS 2 command starts with the `ros2` keyword, followed by a command, a verb, and optionally, positional/optional arguments.
- **General command format:**
  - `ros2 [command] [verb] <positional-argument> <optional-arguments>`
- **For help on ROS 2 CLI commands:**
  - `$ ros2 [command] --help` - Displays help for the command.
  - `$ ros2 [command] [verb] -h` - Displays help for a specific verb.

Action

- **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.
- **Action Command format:**
  - `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

- **Bag:** A file format used to record and playback ROS 2 topics.
- **Command format:**
  - `ros2 bag [verb] <arguments>`
- **Verbs:**
  - `record <topic_name>`: Record the data published to the topic.
  - `info <bag_file_name>`: Get details about the bag file.
  - `play <bag_file_name>`: Replay 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`: Record all topics, required if no topics are listed explicitly or through a regex.
  - `-e`: Record only topics matching the provided regular expression.
  - `-x`: Exclude topics matching the provided regular expression.
  - `-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

- **Component:** A modular unit of software that encapsulates functionality, data, and communication.
- **ROS 2 Components ≈ ROS 1 Nodelets**
- **Command format:**
  - `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 in 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.

Yassine Cherni

  - **--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

- **Control:** A control framework to simplify integrating new hardware and overcome some drawbacks.
- **Command format:**
  - **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 hardware interfaces, 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 from a controller manager.
  - **view_controller_chains**: Generate a diagram of the loaded chained controllers.
- **Arguments:**
  - **-c**: Name of the controller manager ROS node.
  - **--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

- **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 format:**
  - **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

- **Doctor:** Checks all aspects of ROS 2, and warns about possible errors and reasons for issues.
- **Command format:**
  - **ros2 doctor <arguments>** - where `<arguments>` can be various options like `--report`, `-rf`, and `-iw`.
- **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 Points

- **Extension Points:** Lists extension points.
- **Command format:**
  - **ros2 extension_points <arguments>** - where `<arguments>` can include `--all`, `-a`, `--verbose`, and `-v`.
- **Arguments:**
  - **--all, -a**: Show extension points which failed to be imported.
  - **--verbose, -v**: Show more information for each extension point.
- **Examples:**

Yassine Cherni

  - `$ ros2 extension_points`
  - `$ ros2 extension_points --all`

Extensions

- **Extensions:** Lists extensions of a package.
- **Command format:**
  - **`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`

Interface

- **Interface:** ROS applications typically communicate through interfaces of one of three types: messages, services, and actions.
- **Command format:**
  - **`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

- **Launch File:** Allows executing multiple nodes with their own complete configuration (remapping, parameters, etc.) in a single file, which can be launched with only one command line.
- **Command format:**
  - **`ros2 launch [package_name] [launch_file_name] <arguments>`**
- **Arguments:**
  - **`-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` environment variable.
  - **`--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

- **Lifecycle:** Manages nodes 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 format:**
  - **`ros2 lifecycle [verb]`**
- **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

- **Multicast:** In order to communicate successfully via DDS, the used network interface has to be multicast-enabled.
- **Command format:**
  - **`ros2 multicast [verb]`**

ROS 2 Commands Cheat Sheet

Yassine Cherni

- **Verbs:**
  - **`receive`:** Receive a single UDP multicast packet.
  - **`send`:** Send a single UDP multicast packet.
- **Examples:**
  - `$ ros2 multicast receive`
  - `$ ros2 multicast send`

Node

- **Node:** An executable within a ROS 2 package that performs computation and uses client libraries to communicate with other nodes.
- **Command format for running a node:**
  - **`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' names while executing a node.
- **Examples for running a node:**
  - `$ ros2 run turtlesim turtlesim_node`
  - `$ ros2 run turtlesim turtlesim_node --ros-args --remap __node:=my_turtle`
- **Command format for interacting with nodes:**
  - **`ros2 node [command]`**
- **Verbs:**
  - **`list`:** Lists the names of all running nodes.
  - **`info <node_name>`:** Access more information about a specific node.
- **Examples for interacting with nodes:**
  - `$ ros2 node info /my_turtle`

Package

- **Package:** An organizational unit for ROS 2 code that promotes software reuse.
- **Create a ROS 2 package:**
  - **For a Python package:**
    * `$ cd <workspace_name>/src`
    * `$ ros2 pkg create --build-type ament_python [package_name]`
  - **For a C++ package:**
    * `$ cd <workspace_name>/src`
    * `$ ros2 pkg create --build-type ament_cmake [package_name]`

Parameter

- **Parameter:** A list of configuration values attached to a node.
- **Command format:**
  - **`ros2 param [verb] <arguments>`**
- **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.
  - **`param dump <node_name>`:** View a node's current parameter values.
  - **`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

- **rqt tools:** The rqt tools allow graphical representations of ROS nodes, topics, messages, and other information.
- **Command format:**
  - **`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.
- **Examples:**
  - `$ rqt_graph`
  - `$ rqt`

Security

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

ROS 2 Commands Cheat Sheet

Yassine Cherni

  - **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

- **Service:** Communication-based on a call-and-response model, services only provide data when they are specifically called by a client.
- **Command format:**
  - **ros2 service [verb] <arguments>**
- **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>**: 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

- **Topic:** Element of the ROS graph that acts as a bus for nodes to exchange messages.
- **Command format:**
  - **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 message.
- **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

- **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`

Colcon Tools

- **colcon:** 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.
- **Command format:**
  - **colcon [verb] <argument>**
- **Help on colcon CLI commands:**
  - `$ colcon --help`
  - `$ colcon [verb] -h`

ROS 2 Commands Cheat Sheet

Yassine Cherni

colcon build

- **colcon build:** Build a set of packages.
- **Command format:**
  - `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

- **colcon graph:** Generates a visual representation of the package dependency graph.
- **Command format:**
  - `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`

colcon info

- **colcon info:** Shows detailed information about packages.
- **Command format:**
  - `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

- **colcon list:** Enumerates a set of packages.
- **Command format:**
  - `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`

Yassine Cherni

  - `$ colcon list --paths-only`

colcon test

- **colcon test:** Runs the tests for a set of packages.
- **Command format:**
  - `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`

colcon test-result

- **colcon test-result:** Summarizes the results of previously run tests.
- **Command format:**
  - `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`