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:

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

Global options:

- --log-base <path> The base path for all log directories (default: log).
- --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.

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