colcon

Universal Build Tool

Nov. 1st 2019, Dirk Thomas
ROSCon 2019, Macau, China

open robotics

Building a package

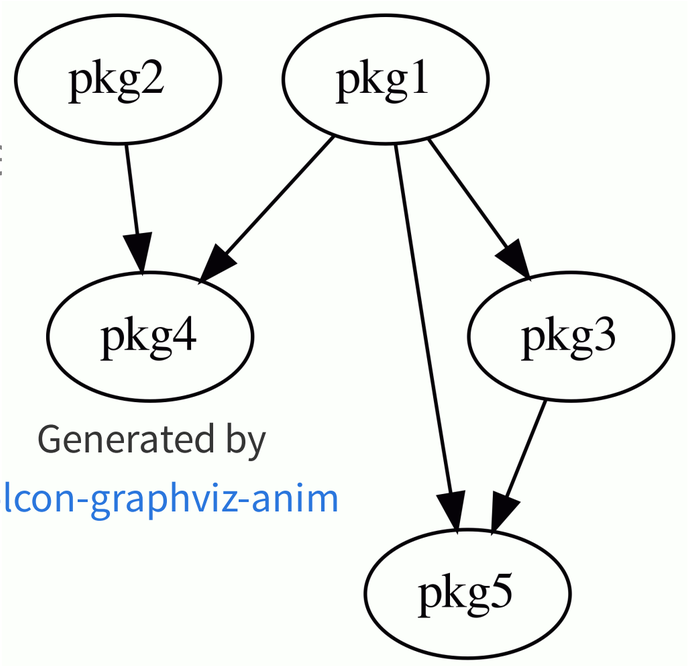
- Prerequisites:
  - all external dependencies are available
  - the environment is set up to make ext. dep. locatable
- **Input:** the sources of a package
- Build the package using its *build system*
  - Install the package artifacts
- **Output:** artifacts ready to be used
  - Setup the environment to use the installed package

No, we don't want to read the `README` to figure out:

- what dependencies are needed
- what build system is being used and how to invoke it
- what environment variables need to be updated before / after

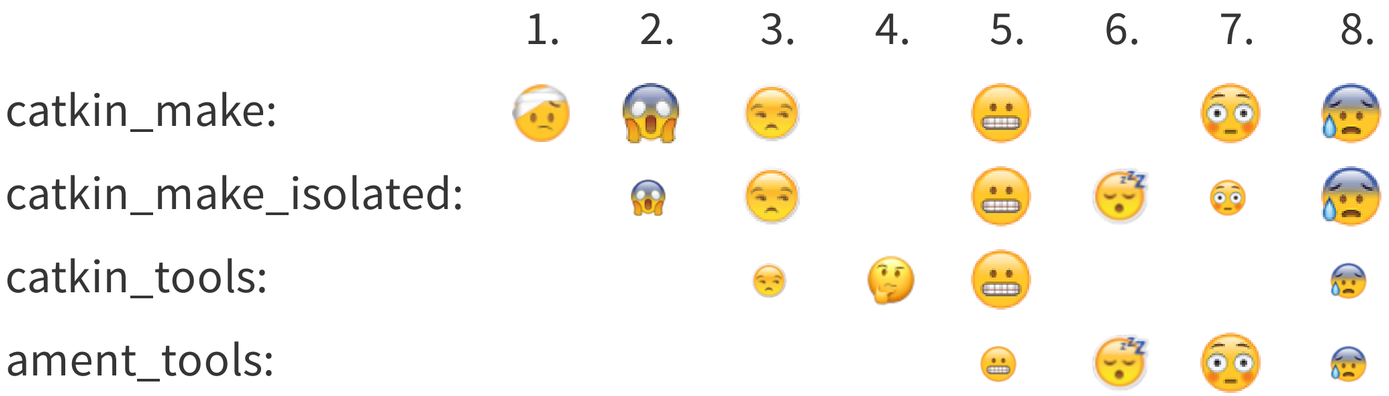
Building a set of packages

- Prerequisites:
  - all external dependencies are available
  - the environment is set up
    to make ext. dep. locatable
- **Input:** the sources of a set of packages
- Compute the dependency graph
- Process the packages in topological order:
  for each package
  - Update the environment to include previous built packages
    which the package depends on
  - Build and install the package using its *build system*
- **Output:** artifacts ready to be used
  - Setup the environment to use all installed packages



Generated by
colcon-graphviz-anim

Wait, don't we have that already?



But...

1. A pkg shouldn't need to know about the internals of other pkgs
2. Should be deterministic - unrelated pkgs shouldn't affect the result
3. Ability to process various package types, e.g. CMake, Python, <put-your-here>
4. Native support for Linux, macOS, and Windows
5. Support packages "as-is", e.g. no `package.xml` as for FastRTPS
6. Build as fast as possible
7. Good user experience / usability features
8. Extensible code base

collective construction

High level goals

- The tool should make building, testing and using multiple packages easy
- It should be possible to add support for
  any kind of build system using extensions
- It should be possible to build any set of packages without requiring changes to their sources - if necessary missing information can be provided externally,
  e.g.:
  - Build Ignition packages
  - Handle one-off cases, e.g. pass `-DENABLE_FOO` to a single package
- After building packages they must be immediately usable
  which includes setting up necessary environment variables etc.

From colcon.readthedocs.io

open
robotics

collective construction

Out of scope - covered by other tools

- Fetch sources of the packages → `vcstool`
- Install dependencies of the packages → `rosdep`
- Create pkg-level binary packages (e.g. `.deb`) → `bloom`, `dpkg-buildpackage`

From colcon.readthedocs.io

open robotics

collective construction

Software engineering goals

- All the functionality provided should be exposed in a way that it can be reused by other extensions
- The separation into multiple Python packages is being used to encourage modularity and loose coupling (Law of Demeter), it is also used to demonstrate extensibility and show that certain features are not "special" but can be contributed externally
- Each component should have responsibility over a single part of the software (Single responsibility principle)
- Each functionality added should follow the principle "you don’t pay for what you don’t use"

From colcon.readthedocs.io

open robotics

Setup colcon

- Install `colcon` including common extensions, via:
  - the Debian package `python3-colcon-common-extensions` or
  - the Python package `colcon-common-extensions`

- Using `bash` or `zsh`? Get completion!
  - Install `python3-argcomplete` [deb] / `argcomplete` [pip]
  - Source the completion script `colcon-argcomplete.bash|zsh` from:
    - `/usr/share/colcon_argcomplete/hook/` or
    - `$HOME/.local/share/colcon_argcomplete/hook/`

- Liked `roscd`? Try `colcon_cd`
  - Source the script `colcon_cd.sh` (similar location)

- Check for updates in the future: `colcon version-check`

For more details see colcon.readthedocs.io

Workspace Layout

In ROS the sources of the packages to be processes are commonly placed

- within a workspace root `<ws>`
- in a subdirectory named `src`

```
<ws>
|-- src
    |-- dir1
        |-- package.xml [with the name tag containing "pkg1"]
        |-- some_name
            |-- package.xml [nested under another package not supported]
    |-- dir2
        |-- dir3
            |-- CMakeLists.txt [with the function call "project(pkg3)"]
        |-- dir4
            |-- setup.py [with the setup argument "name='pkg4'"]
    |-- dir5
        |-- COLCON_IGNORE [empty marker file]
        |-- ...
```

`build` Command

- Build all packages in the workspace: `colcon build`
  - Minimal output showing progress on a by-package level (if everything goes smooth)
  - By default if any package fails all ongoing packages are aborted, no others are started
  - E.g. pass custom CMake args: `--cmake-args` which are applied to all processed packages

Note: if you build more than once make sure to use `ccache`.

```
$ colcon build
Starting >>> pkg1
Starting >>> pkg2
Finished <<< pkg1 [10s]
Starting >>> pkg3
Finished <<< pkg3 [10s]
Starting >>> pkg5
Finished <<< pkg2 [25s]
Starting >>> pkg4
Finished <<< pkg5 [20s]
Finished <<< pkg4 [20s]

Summary: 5 packages finished [45s]
```

Metadata

- Sometimes you want to pass pkg-specific arguments
  - Via CLI that would be ~~cumbersome~~ verbose
  - Therefore being done via `yaml` files
    - A file named `./colcon.meta` is picked up automatically
    - Otherwise pass custom files using `--metas`

```
{
  "names": {
    "fastrtps": {
      "cmake-args": ["-DSECURITY=ON"]
    }
  }
}
```

See colcon.readthedocs.io for information about `.meta` files.
See colcon-metadata-repository for information how to share such files.

Event Handlers

- By default the actual output of the build isn't shown.
  - Why? Because otherwise the output from concurrent packages would interleave.
- Instead:
  - Status line

```
[23s] [2/5 complete] [2 ongoing] [pkg2:install - 23s] [pkg5:cmake - 1s]
```

  - Any `stderr` output will be shown after a package finished
    - `stderr` output doesn't mean it failed
    - it could fail without `stderr` output (e.g. on Windows)
  - But - also by default - any output it written to a `log` file

```
<ws>
|-- log
    |-- <cmd>_<timestamp>
    |-- latest_<cmd>
    |-- latest
        |-- events.log
        |-- logger_all.log
        |-- <pkg>
            |-- command.log
            |-- stderr.log
            |-- stdout.log
            |-- stdout_stderr.log
            |-- streams.log
```

- See section **Event handler arguments** in `colcon build --help`
  - Show all output after a pkg finished: `--event-handlers console_cohesion+`
  - Output on-the-fly (usually only when building a single pkg): `console_direct+`
  - The suffix `+` enables a specific handler, `-` disables it

`list` Command

1. **Discovery**

   Determine directories to check for a package
   - Default: recursively crawl for pkgs `--base-paths .`
   - Alternatives:
     - explicit (non-recursive) list of paths `--paths`
     - get a list of paths from a configuration file `--metas`
   - See section **Discovery arguments** in `colcon list --help` for all arguments
2. **Identification**

   Determine if a directory contains a package
   as well as its name and type, e.g.:
   - ROS pkg: has a `package.xml` following a specific schema
   - CMake pkg: has a `CMakeLists.txt` file
   - Python pkg: has a `setup.py` file
3. **Augmentation**

   Add metadata to identified packages
   - E.g.: a package named `Gazebo` has a `share/gazebo/setup.sh` file which should be sourced (see colcon-metadata-repository)

```
<ws>
|-- src
    |-- dir1
        |-- package.xml
        |-- some_name
            |-- package.xml
    |-- dir2
        |-- dir3
            |-- CMakeLists.txt
        |-- dir4
            |-- setup.py
    |-- dir5
        |-- COLCON_IGNORE
        |-- ...
```

```
$ colcon list
pkg1   <ws>/src/dir1         (ros.ament_cmake)
pkg3   <ws>/src/dir2/dir3    (cmake)
pkg4   <ws>/src/dir2/dir4    (python)
```

`info` Command

```
$ colcon info ament_cmake_core
path: <ws>/src/ament_cmake_core
    type: ros.ament_cmake
    name: ament_cmake_core
    dependencies:
      build: ament_package cmake python3-catkin-pkg-modules
      run: ament_package cmake python3-catkin-pkg-modules
      test:
    metadata:
      version: 0.8.0
```

- The `ros.` prefix identifies that the package has a `package.xml` file
- Together with the `ament_cmake` suffix it determines how the package is processed

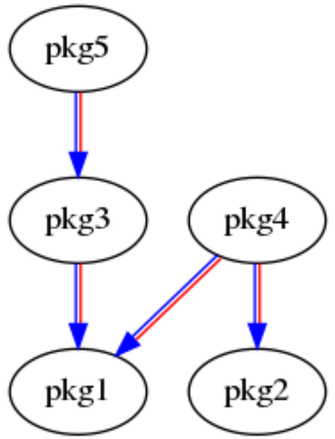
- The dependencies are (in this case) extracted from the `package.xml` file
- Some dependencies are package names in the workspace, others name external dependencies (those are irrelevant for colcon)

`graph` Command

Visualize the dependencies between packages

```
$ colcon graph --legend
+ marks when the package in this row can be processed
* marks a direct dependency from the package indicated
  by the + in the same column to the package in this row
. marks a transitive dependency

pkg1   + **.
pkg2    + *
pkg3     + *
pkg4      +
pkg5       +
```



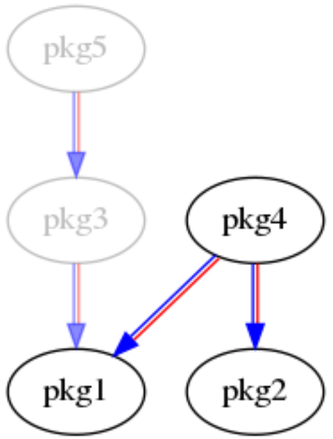
`colcon graph --dot | dot ...`

- Processing the packages sequentially from top to bottom ensures the topological order
- Pkgs without dependencies between each other can be processed concurrently
  - The order of `pkg4` and `pkg5` depends on which dependencies are completed first

open robotics

Package Selection Arguments

Instead of processing all packages in a workspace you often want to only process a subset to speed up your workflow.



The following options apply to various commands:

- After making changes to packages:
  - Only process specific packages:
    `--packages-select <pkgname1> [<pkgname2> ..]`
  - *Note: when used with `build` their dependencies must have been build before*
- Don't know what has changed and only want to use `pkgX`?
  - Build up to specific pkgs - including recursive dependencies: `--packages-up-to pkgX`
- Process all recursive downstream packages of `pkgY`?
  - Including `pkgX`: `--packages-above <pkgY>`
  - Excluding `pkgX`: `--packages-select-by-dep <pkgY>`

- See section **Package selection arguments** in e.g. `colcon build --help`
  - E.g. `--packages-select-regex <regex>`, `--packages-select-build-failed`

Note: there are options to `skip` packages. Skipping is different from previous seen ignoring. The former doesn't process the package but uses it as a dependency for others. The later is equal to the package not being in the workspace.

More Complex Package Selection

Sometimes the existing package selection arguments
aren't covering what you want to select / skip.

1. Combined logic using nested invocations:
   - E.g. use `--packages-up-to` but with multiple package names matching a regular expression:
     - Use `colcon list --packages-select-regex <regex>` to determine the set of packages
     - Use `--packages-up-to` with the result of the previous determined package names
   - `colcon build --packages-up-to `colcon list --packages-select-regex <regex>``

2. Write your own package selection option:
   - Any Python package can contribute an extension providing additional command line arguments

`test` / `test-result` Command

- Test all packages in the workspace: `colcon test`
  - *Note: must `build` a package before testing*
  - E.g. pass custom CTest / pytest args: `--ctest-args` / `--pytest-args`
  - Dealing with flaky tests:
    - Identify flaky tests: `--retest-until-fail N`
    - Get flaky tests to pass: `--retest-until-pass N`
  - Package Selection arguments:
    - After making changes to a package `build` and `test` the package itself as well as all downstream packages: `--packages-above`
    - Retesting pkgs with test failures in previous runs: `--packages-select-test-failures`
  - Failing tests by default still result in a return code of zero (meaning the tests were successfully run)
    - To change that pass `--return-code-on-test-failure`

- Get a summary of all failed tests: `colcon test-result`
  - Show additional information about failed tests: `--verbose`
  - Start fresh, remove previous test results: `--delete`

Pointers

- `colcon --help` / `colcon <command> --help`
- colcon.readthedocs.io
- For more context: ROS 2 Design Article
- How to contribute, improvements, bug fixes and documentation
    - Already many contributed extensions: bazel, bundle, cargo, gradle, lcov-result, sanitizer-reports, spawn-shell, ...
- Use it with ROS 2, ROS 1, Gazebo, Ingition, ...

Questions...

open robotics