ROS Catkin UX Discussion: Improving Catkin and its Tools

Planning for June 9th, 10:15am (PDT) 2014 Video Conference

Participants (add your name here and speak your mind!):

- Jonathan Bohren
- Mike Purvis
- William Woodall
- Dirk Thomas
- Jack O'Quin

Please add any concerns with the current released catkin toolchain or enhancement ideas for future versions in Sections 2 and 3 respectively. Additionally, please add your comments under each concern or enhancement in a flat list, responses and discussion of each person's comments will happen during the videoconference. These comments should be questions, related concerns, or even proposed solutions.

Table Of Contents

Table Of Contents
1. Non-Issues with the Current Released Toolchain
2. Concerns with the Current Released Toolchain
    2.1. "Auto-Extension" and "Rollback" of Workspace Environment Is Counter-Intuitive
        Proposal / Jonathan Bohren (catkin_tools#58)
        Proposal / Jonathan Bohren
    2.2. Lack of Introspection into Workspace Configuration
        Proposal / Jonathan Bohren
    2.3. Configure-Time Cross-Talk
        Proposal / Jonathan Bohren
    2.4. Overloaded use of `$CMAKE_PREFIX_PATH`
        Proposal / Jonathan Bohren (based on comments in catkin_tools#47)
    2.5. Workspace Management In General
        Proposal / Jonathan Bohren
        Question / Jack O'Quin
    2.6. Nomenclature: What is a "Catkin Workspace" ?
        Comment / Jonathan Bohren
    2.7. Novice User Failure Modes
        Proposal / Jonathan Bohren
    2.8. Deprecation Pathway For catkin_make and catkin_make_isolated
        Proposal / Jonathan Bohren
        Question / Mike Purvis
        Question / Opinion / Vincent Rabaud

Comment / Jack O'Quin
2.9. There is No Well-Defined Pattern / API for Cleaning Up Catkin env-hooks
Proposal / Jonathan Bohren
Question / Mike Purvis
Question / Jack O'Quin
3. Desired Functionality and Interface Enhancements
3.1. Initiating Workspace Builds Without Navigating to Workspace Root
Proposal / Jonathan Bohren
3.2. Building Pure CMake Projects Without Adding a package.xml
Proposal / Jonathan Bohren
4. General remarks
4.1 Targeted ROS distros
4.2 Tools on top of core functionality
4.3 Environments and CMake
Conclusions / follow ups

1. Non-Issues with the Current Released Toolchain

The following are considered "acceptable" and are not the focus of this discussion:

- CMake API
- package.xml format (v2)

2. Concerns with the Current Released Toolchain

This is a summary of several of the concerns raised in relation to catkin_tools#48. Most of the concerns relevant to this discussion are related to workspace configuration, management, and debugging. This discussion should have an emphasis on streamlining flexible development, introspection, and design with novice users in mind.

The numbered sub-sections of this section each pertain to a given concern about the catkin toolchain user experience and workflow. In each section is a bulleted list of solutions or comments by a user.

2.1. "Auto-Extension" and "Rollback" of Workspace Environment Is Counter-Intuitive

Currently, when a user invokes `catkin build`, the tool assumes that the user wants to build the current workspace against all paths listed in the `$CMAKE_PREFIX_PATH` environment variable. This assumption is correct only inasmuch as the user knows what is already in his or her `$CMAKE_PREFIX_PATH`.

If one understands the underlying mechanism that Catkin uses, it's easy to predict the results of actions:

Given the following actions:

```
# Load /opt/ros/hydro workspace
source /opt/ros/hydro/setup.bash
# Make a new workspace, called "ws1"
mkdir -p ~/ws1/src
cd ~/ws1/src && catkin_create_pkg foo
# Build the new workspace, chaining it from /opt/ros/hydro
cd ~/ws1 && catkin_make
```

Catkin has the following "sticky" behavior:

- The initial configure-time CMAKE_PREFIX_PATH, "/opt/ros/hydro", is statically stored in the following places:
    - ~/ws1/build/catkin_generated/setup_cached.sh
    - ~/ws1/build/catkin_generated/generate_cached_setup.py
    - ~/ws1/devel/_setup_util.py
- The only way to fully re-set the parent workspace is by removing the build directory, re-setting the environment, and re-building. Simply changing the cached CMAKE_PREFIX_PATH is not enough.
- Without re-setting the workspace, it will always be chained against the CMAKE_PREFIX_PATH present when the package was originally configured

In an informal poll of several students a JHU who have been using ROS proficiently for the better part of a year, not one could answer the following questions correctly. Furthermore each found the answers to be surprising, even though they said ji "made sense" after explaining how catkin worked and how catkin stored $CMAKE_PREFIX_PATH:

- *What is the value of $CMAKE_PREFIX_PATH after the following sequence of actions?*

```
unset CMAKE_PREFIX_PATH
source /opt/ros/hydro/setup.bash
mkdir -p ~/ws1/src
# add packages to ws1/src
cd ~/ws1
catkin_make
source devel/setup.bash
source /opt/ros/hydro/setup.bash
```

Highlight options to see the solution

A. "/home/user/ws1/devel:/opt/ros/hydro"

B. "/opt/ros/hydro:/home/user/ws1/devel:/opt/ros/hydro"
C. "/opt/ros/hydro"

- *What is the value of $CMAKE_PREFIX_PATH after the following sequence of actions?*

```
unset CMAKE_PREFIX_PATH
source /opt/ros/hydro/setup.bash
mkdir -p ~/ws1/src
# add packages to ws1/src
cd ~/ws1
catkin_make
source devel/setup.bash
mkdir -p ~/ws2/src
# add packages to ws2/src
cd ~/ws2
catkin_make
source /opt/ros/hydro/setup.bash
catkin_make
source devel/setup.bash
```

Highlight options to see the solution

A. "/home/user/ws2/devel:/home/user/ws1/devel:/opt/ros/hydro"
B. "/home/user/ws2/devel:/opt/ros/hydro"
C. "/opt/ros/hydro"

- **Proposal / Jonathan Bohren (catkin_tools#58)**
  The automatic chaining is definitely convenient, and there are good arguments against requiring people to explicitly specify which workspace(s) they want to chain their new workspace against.

  I've written a patch (catkin_tools#58) for `catkin build` which adds an `--extend-ws` argument which lets a user explicitly specify the workspace that a new (or existing) workspace should be chained against. This is based on discussions from catkin#643. This is implemented by sourcing the setup-file from the workspace in question and extracting the resulting environment variables.

  Additionally, `catkin build` could add the notion of a "default base workspace" which could either be whatever's in `CMAKE_PREFIX_PATH` (the current behavior), or it could be something like /opt/ros/hydro or ~/path/to/my/overlay/devel. This would make it even less likely that people accidentally chain workspaces together which they didn't intend to.

- **Proposal / Jonathan Bohren**
  In order to increase visibility of catkin's modification of the `CMAKE_PREFIX_PATH` when sourcing environments, any changes to this variable could be announced when sourcing one of the setup files. For example, if an element in `CMAKE_PREFIX_PATH` is getting "rolled back" then the setup file should announce that to stdout. Even in the nominal case where a `CMAKE_PREFIX_PATH` is just extended, the setup file should make it clear to the user what is happening. Of course it should be possible to suppress these messages with a `--quiet` flag.

2.2. Lack of Introspection into Workspace Configuration

Once a workspace has been created, it's hard to determine which settings were used to build it. Such settings include:

- build/devel/src/install-space paths
- additional CMake arguments
- build "style" merged/isolated
- devel "style" merged/isolated

- **Proposal / Jonathan Bohren**
  Recently, `catkin_make/_isolated` now generate an empty `.catkin_workspace` file in the root of a given workspace. Instead of leaving this file empty, catkin could put configuration information in this file which could specify all of the above settings.

  This solution prompts two concerns that would need to be resolved:
  1. What happens when the user edits the file? Does it throw an error or does it just switch the behavior in the same way that modifying .git/config does?
  2. What happens when the user wants to clean out the repository. Currently this can be done confidently by removing the buildspace and develspace directories. We would need to detect that the .catkin_workspace file describes non-existent directories and then either ignore the options, or alternatively assume that the user wants to perform the same options. Either way, an appropriate amount of feedback would need to be given to the user.

2.3. Configure-Time Cross-Talk

Catkin's merged-build behavior has classically suffered from both bugs and un-bugs that result from the side-effects of configuring one package before another. These include, but are not limited to:

- modifications of environment variables,
- global cmake variables,

- target name collisions,
- failing to call `catkin_package()` before declaring targets

Additionally, the modification of a single CMakeLists.txt file prompts re-configuration of every package in a workspace, which puts a high cost on large workspaces.

- **Proposal / Jonathan Bohren**
  Use `catkin build`, which by default builds each package in isolation while depositing build products into a common devel-space.

2.4. Overloaded use of `$CMAKE_PREFIX_PATH`

Catkin uses the $CMAKE_PREFIX_PATH environment variable to manage the list of "sourced" workspaces. This is already a confusing point for new users. When using CMake directly, it's understandable that they would have to manipulate $CMAKE_PREFIX_PATH, but when using tools like catkin_make and catkin build, students and novice users are confused that to reset their *catkin* environment they need to unset $CMAKE_PREFIX_PATH.

- **Proposal / Jonathan Bohren (based on comments in catkin_tools#47)**

  We could switch to using a catkin-specific environment variable, call it $CATKIN_PREFIX_PATH which is added to $CMAKE_PREFIX_PATH either when you call find_package(catkin ...) or when you run catkin build. Catkin can add and remove whatever it wants to $CATKIN_PREFIX_PATH, without concern for colliding with users' own modifications to $CMAKE_PREFIX_PATH and the new variable will be more intuitive for novices trying to understand the buildsystem.

  It makes things more straightforward in three ways:

  1. It means that we can be more strict about how we interpret $CATKIN_PREFIX_PATH since only catkin should be modifying it, and a path on $CATKIN_PREFIX_PATH must be a catkin workspace. This strictness means we can more easily detect a broken environment automatically.
  2. It means that a user who knows about environment variables can more easily associate $CATKIN_PREFIX_PATH with catkin when trying to debug their environment.
  3. It means that people can add things to their $CMAKE_PREFIX_PATH without having to worry about it colliding with catkin use of the same variable. This is important on less-supported platforms like OS X and other systems where there might be non-standard install paths for libraries.

2.5. Workspace Management In General

Currently, due to the lack of introspection into workspace configurations, and a lack of confidence of novice users for manipulating `CMAKE_PREFIX_PATH`, it is hard to confidently switch between workspaces as well as re-define the settings for a given workspace.

- ### Proposal / Jonathan Bohren
  We could add a new `catkin workspace` or `catkin ws` verb to catkin_tools which can be used to both name workspaces in a globally-identifiable way as well as perform the following workspace management procedures:

  - `catkin ws create [<workspace-name>]`
    Create a new workspace with default directories and an optional identifier
  - `catkin ws create --extend <workspace-name>`
    Create a new workspace that explicitly extends another workspace either by name or by path
  - `catkin ws save [--default] <workspace-name>`
    Save the current workspace to a persistent file with an identifier, or set an already saved workspace as the default.
  - `catkin ws get <workspace-name>`
    Print the path to the workspace identified by <workspace-name>
  - `catkin ws load [<workspace-name>]`
    Load either the default or a named workspace environment from a persistent file. This could go into your shell profile so each new shell gets the workspace that you're currently using.
  - `catkin ws list`
    List the saved workspaces
  - `catkin ws clean`
    Remove the appropriate build and devel directories (prevents people from having to use rm -rf in their workspaces
  - `catkin ws info <workspace-name>`
    Show a workspace's dependencies, which known workspaces depend on it, number of packages, if it's been built, how it’s been built, etc
  - `catkin ws discover <path>`
    Find all catkin workspaces under some path (by looking for marker file introduced here ros-infrastructure/catkin_pkg#95)

  For workspace names, there could be some defaults like ros-hydro for /opt/ros/hydro but people will also use project names for different workspaces.

The persistent file could look something like ~/.config/catkin/workspaces.yaml:

```
workspaces:
  hydro: '/opt/ros/hydro'
  indigo: '/opt/ros/indigo'
  overlay: '/home/jbohren/ws/overlay/devel'
  jhu: '/home/jbohren/ws/jhu/devel'
  nasa: '/home/jbohren/ITAR/nasa/devel'
```

A catkin ws command should be concerned solely with managing catkin workspaces as they pertain to the catkin buildsystem. It shouldn't be concerned with rosdep, wstool, or other external tools. This verb would also help with workspace introspection.

- **Question / Jack O'Quin**
  Should most of these commands should assume current workspace by default?

2.6. Nomenclature: What is a "Catkin Workspace" ?

The term "catkin workspace" has been used to describe several semantically-different directories in a user's filesystem. It would be great to be able to be consistent across all tools related to catkin and ROS. Which of the following is correct:

1. A directory from which `catkin_make{_isolated}` has been called
2. A directory containing a `.catkin` marker file
3. A directory containing a `.catkin_workspace` marker file
4. The "source-space" used in a catkin build
5. The "devel-space" used in a catkin build
6. The "install-space" used in a catkin build
7. A directory containing ROS/Catkin setup files

- **Comment / William Woodall**
  This is defined here, if not sufficient then we should update that REP:
  http://www.ros.org/reps/rep-0128.html

Furthermore, other ROS and ROS-related tools define other kinds of workspaces, as well:

- `rosws` both manages VCS checkouts and setup files
- `wstool` manages VCS checkouts in a source-space

Other nomenclature questions:

- What does it mean to "clean" a catkin workspace? How do you "start over from scratch"?

- When you build one workspace after building another and adding it to your CMAKE_PREFIX_PATH, is it called "chaining" or "extending"? In the latter, which workspace is the one "being extended"?
- Are chained workspaces "parent" and "child" workspaces?

Note that a partial Catkin glossary is given here: http://wiki.ros.org/catkin/Glossary

---

- **Comment / Jonathan Bohren**
  I think it would be ideal if we could unify the "workspace" moniker so that we could talk about catkin and wstool in the same sentence without having to use extra adjectives for describing different kinds of workspaces. ROS development is dramatically enhanced by the use of wstool, yet, since its forking from rosinstall, it has been pushed back into the shadows.
- **Comment / William Woodall**
  I would disagree with the assertion that wstool is being pushed back into the shadows, because it is used in all of our instructions, except in the very lowest tutorials where we are showing the bare minimum required to build a catkin workspace. wstool is not required to build a workspace (which I think is a good thing) but is recommended and used in any tutorials or instructions I have a part in, e.g. the source install instructions and any posts I make to answers.ros.org.

2.7. Novice User Failure Modes

Novice users have been observed to have most difficulty understanding the hidden behaviors of catkin like workspace chaining which can be done unintentionally. A recently-patched bug (catkin#641) also prevented users from cleaning their workspaces and starting over from what they believed was a "clean" workspace.

Due to the broad spectrum of use cases that catkin is designed to satisfy, there is a LOT of documentation:

- http://wiki.ros.org/catkin
- http://wiki.ros.org/catkin/conceptual_overview
- http://docs.ros.org/api/catkin/html/
- http://wiki.ros.org/catkin/package.xml
- http://wiki.ros.org/catkin/CMakeLists.txt
- http://wiki.ros.org/catkin/Glossary
- http://www.ros.org/reps/rep-0140.html
- http://wiki.ros.org/catkin/what

There is additionally more documentation for `catkin build` and other functionality in catkin_tools.

As such, it is unlikely that a user will actually *read* all of this documentation, and novice users will always miss something. They might not even know that they're missing something.

- **Proposal / Jonathan Bohren**
  Add a pedantic / tutor mode for `catkin build` that can be set by default when it is installed, and toggled easily with a catkin verb like `catkin --enable-tutor` or `catkin --disable-tutor`. This global mode will have more verbose information and spend more time looking for user errors.

2.8. Deprecation Pathway For catkin_make and catkin_make_isolated

Since `catkin build` is planned to supercede `catkin_make` and `cakin_make_isolated`, there is a huge corpus of tutorials which need to be updated. What is the strategy for rolling out this new tool?

- **Proposal / Jonathan Bohren**
  We could simply create wrapper scripts called `catkin_make` and `catkin_make_isolated` which call through to `catkin build` and then migrate the documentation as slowly as we need to.

- **Question / Mike Purvis**
  A previous discussion gave me the impression `catkin build` would become recommended, but that `catkin_make` would live on. Is that still the case? Either way, what's the current and future policy avoiding target name collisions? (eg, necessity of prefixing target names with package names?)

- **Question / Opinion / Vincent Rabaud**
  I have rtfm-ed "catkin build" but it seems like there is no option to have one common build space. How do we deal with cross targets then ? (like docs/tests)
  I also dislike the fact that I have to install yet another tool to use catkin (you now need to install catkin_pkg and catkin to use a catkin package). Having catkin self-contained is highly desirable to me: something that just does the symlink, cmake, make: it is more transparent what is happening.
- **Comment / William Woodall**
  I believe that we have already discussed this in the mailing list, `catkin_make` will live on, as is, so will the ability to just call cmake on a source space with the toplevel.cmake symlinked in there, which is how you get a combined build space. `catkin build` will be recommended for users to avoid issues like colliding target names, inter-target ordering problems, and other problems people have pointed out with plain `catkin_make`. We discussed replacing the implementation of `catkin_make_isolated` with a shell that converts the arguments and then calls `catkin build` as well as displaying a warning the that user should switch to `catkin build`. `catkin build` will not provide a combined build

space option, because they are orthogonal styles of building a workspace.

w.r.t to “How do we deal with cross targets then ? (like docs/tests)”, I’m not sure what you mean, can you be more specific?

“I also dislike the fact that I have to install yet another tool to use catkin” I agree, and catkin_tools is strictly _not_ required to use catkin in the simplest case. I have made the suggestion to embed `catkin_pkg` into `catkin` so that it is not required to be installed to use catkin, but I have met resistance to this idea, and I have to admit that I am not 100% certain that is would be better that way.

- **Comment / Jack O’Quin**
  One of the conveniences of catkin_make workspaces was:
  ```
      catkin_make
      cd build
      make roslint run_tests
  ```
  or:
  ```
      make run_tests_camera1394
  ```
  I have not yet discovered a good way to do either with catkin build, except this single-package trick:
  ```
      catkin build camera1394
      (cd build/camera1394/; make run_tests)
  ```
  Making catkin build work from within the source or build spaces would help some:
  ```
      cd build/camera1394
      catkin build camera1394
      make run_tests
      catkin_test_results test_results
  ```

2.9. There is No Well-Defined Pattern / API for Cleaning Up Catkin env-hooks

Catkin currently provides a mechanism through which packages can inject their own shell code into the setup files which get loaded when someone loads a workspace. This is done through the catkin_add_env_hooks() CMake macro.

These hooks are normally used for the following:
- Append directories to a PATH-type environment variable
- Define CLI autocomplete rules
- Define additional shell functions
- Whatever nasty possibly-platform-specific thing someone can think of...

As mentioned in catkin_tools#58, this presents a problem with cleanup after sourcing one or more workspaces. This is especially a problem if someone wishes to source a given catkin workspace in their shell's rc-file, and isn't aware of something that a package in their workspace path is doing. While catkin provides a "rollback" behavior for `$CMAKE_PREFIX_PATH`, there is no additional support to developers for undoing their own changes related to these setup files.

Here is a list of commonly-used patterns in env-hooks (via github search):

- roslisp
  - exports a package-specific variable

```
export ROSLISP_PACKAGE_DIRECTORY=@CMAKE_INSTALL_PREFIX@/share/common-lisp/ros
```

- orocos_toolchain
  - defines some RUBY variables
  - defines some package-specific variables
  - defines a PATH-like variable, RUBYLIB
  - extends LUA_PATH
  - extends LD_LIBRARY_PATH / DYLD_LIBRARY_PATH for plugins

```
RUBY_VERSION=`ruby --version | awk '{ print $2; }' | sed -e "s/\(.*\..*\)\..*/\1/"`
RUBY_ARCH=`ruby --version | sed -e 's/.*\[\(.*\)\]/\1/'`
export RUBYOPT=-rubygems
export TYPELIB_USE_GCCXML=

envpath=@CMAKE_INSTALL_PREFIX@

if [ `uname -s` = Darwin ]; then
        export
        RUBYLIB=$envpath/lib:$envpath/lib/typelib:$envpath/lib/ruby/${RUBY_VERSION}/${RUBY_ARCH}:$
        envpath/lib/ruby/${RUBY_VERSION}:/Library/Ruby/Gems/${RUBY_VERSION}:\
        /Library/Ruby/Gems/${RUBY_VERSION}/${RUBY_ARCH}:/Library/Ruby/Gems
        export DYLD_LIBRARY_PATH=$envpath/lib/typelib:$envpath/lib/orocos:$DYLD_LIBRARY_PATH
else
        export
        RUBYLIB=$envpath/lib:$envpath/lib/typelib:$envpath/lib/ruby/${RUBY_VERSION}/${RUBY_ARCH}:$
        envpath/lib/ruby/${RUBY_VERSION}:
        export LD_LIBRARY_PATH=$envpath/lib/typelib:$envpath/lib/orocos:$LD_LIBRARY_PATH
fi
if [ "x$LUA_PATH" == "x" ];then
        LUA_PATH=";;;"
fi
export LUA_PATH="$LUA_PATH;$envpath/share/lua/5.1/?.lua"
```

- axcli
  - adds shell completion for a program in this package

```
function _roscomplete_axcli
{
    local arg opts
    COMPREPLY=()
    arg="${COMP_WORDS[COMP_CWORD]}"
    local cword=$COMP_CWORD
    for a in $(seq $((COMP_CWORD-1))); do
        if [ -z "${COMP_WORDS[a]//-*}" ]; then
            ((cword--))
        fi
    done
    local words=(${COMP_WORDS[@]//-*})

    if [[ $cword == 1 ]]; then
        opts=`rostopic list | grep '/goal$' | sed 's,/goal$,,' 2> /dev/null`
        COMPREPLY=($(compgen -W "$opts" -- ${arg}))
    elif [[ $cword == 2 ]]; then
        mtype=`rostopic type ${words[1]}/goal`
        opts=`rosmsg-proto msg 2> /dev/null -s ${mtype:0:-10}Goal`
        if [ 0 -eq $? ]; then
            COMPREPLY="$opts"
        fi
    fi
}

complete -F "_roscomplete_axcli" "axcli"
```

- rFSM
    - extends LUA_PATH

```
#!/bin/sh
if [ "x$LUA_PATH" = "x" ]; then
   LUA_PATH=";"
fi
export LUA_PATH="$LUA_PATH;@CMAKE_INSTALL_PREFIX@/share/lua/5.1/rfsm/?.lua"
```

- pano_py
    - extends LD_LIBRARY_PATH

```
export
LD_LIBRARY_PATH=$LD_LIBRARY_PATH:@(CMAKE_INSTALL_PREFIX)/@(CATKIN_GLOBAL_PYTHON_DESTINATION)
```

- **Proposal / Jonathan Bohren**

  Proper env-hooks should be able to clean themselves up just like Catkin workspaces do. To support this, catkin could expose the PATH-modification and rollback logic into appropriate shell functions which can be called by catkin package env-hooks.

  One of the following methods could be used to specify the cleanup hooks:
    1. An additional CMake macro could be added like `catkin_add_env_cleanup_hook()`
    2. The current macro could be modified to take additional argument

3. The filename conventions could be extended such that an env-hook named “10.my_pkg.sh.installspace.in” has an inverse-operation “10.my_pkg.sh.installspace.cleanup.in” (likewise for *.develspace.* etc)

In this way, packages which extend PATH-type environment variables can remove the appropriate items for a given workspace, packages which define shell functions or aliases can unset them, and other cleanup actions can be taken. If someone is injecting something into the setup files, they should be responsible for cleaning it up.

One argument against this solution is that the cleanup scripts might get removed by a user if the user starts deleting various parts of the workspace. If that happens, then the environment will only persist as long as they don’t open a new shell. Users normally only partially delete workspaces, however, when normal debugging tools fail, and it’s unclear what is going wrong. If catkin improves clarity, then it is far less likely that this happens. Even if a user deletes build and devel directories, “rollback” behavior for env-hooks could be written in a hidden file at the root of the workspace. An environment variable pointing to that file could get defined, and in the event that the user tries to source another workspace, we could detect that the rollback script has been removed, and we can report an error, instructing the user to clean his or her environment and start over in a new shell.

- **Question / Mike Purvis**
  With respect to the “potentially nasty platform-specific things” which are possible, it’d be great to have some formal guidance about appropriate/non-nasty uses of this powerful feature, and perhaps a space to propose and receive feedback on possible concepts. For example, we (Clearpath) are considering a package for our platforms which would provide hooks for setup-time hardware detection of optional equipment, and set environment variables accordingly. (eg, if /dev/imu symlink present, flip an env var which enables that driver in the launch file). It hasn’t been clear to me what the appropriate forum is to try to discuss the kinds of platform-level challenges Clearpath faces at our scale, and how to best leverage Catkin and other ROS tooling to address those needs.
- **Question / Jack O’Quin**
  Will this fix up all the other environment variables that must be modified to include per-workspace paths, e.g.: $ROS_PACKAGE_PATH, $PATH, $PYTHONPATH, $LD_LIBRARY_PATH?

  Fixing $CMAKE_PREFIX_PATH without all those others would not help enough for most users to even notice any improvement.

3. Desired Functionality and Interface Enhancements

3.1. Initiating Workspace Builds Without Navigating to Workspace Root

As described in catkin_tools#10, it would be desirable to have context-aware verbs which could enable building a single package just by navigating to it and executing something like `catkin build` to build the whole tree, or `catkin build --this` to build this package and it's dependencies, or `catkin build --just-this` to build this package and ignore its dependencies. (Note that these could be aliased to other verbs).

- **Proposal / Jonathan Bohren**
  The newly-added `.catkin_workspace` file could do part of the job of designating the root of a catkin workspace. `catkin build` could be designed to determine the workspace root from context.

3.2. Building Pure CMake Projects Without Adding a package.xml

Currently, catkin provides an easy way to develop in an isolated FHS-based workspace, even for non-catkin packages which are designed to be installed to your system. In order to build these, however, package.xml files need to be added to the project roots.

- **Proposal / Jonathan Bohren**
  All directories with a top-level `CMakeLists.txt` file could be built before all other packages. This will allow dependency-less packages to be dropped into a catkin workspace without any effort.
- **Comment / William Woodall**
  I like this idea. Though I think it is very likely that we will have cmake projects which depend on each other, so maybe we would need to a lighter weight way to specify dependencies amongst the cmake projects.

4. General remarks

4.1 Targeted ROS distros

In general it would be great to have a consensus about a solution which could be drafted on a clear plate (without locking us in based on how the current system is designed) and then consider how this can be integrated into the existing architecture.

Depending on the proposed changes we have to consider for which ROS distro those should be realized.

4.2 Tools on top of core functionality
While the tools (catkin build, catkin_make(_isolated)) provide significant usability improvements it must remain possible to build packages without them. So any solution proposed must also apply to plain command sequences (e.g. cmake / make / install / source).

4.3 Environments and CMake
CMake uses the shell environment available at the first configure time to make certain decisions and cache these results in the build/CMakeCache.txt file. Even when the environment is changed some of those decision are not reconsidered.

Conclusions / follow ups

- 2.1 Come up with ways to improve rollback (https://github.com/ros/catkin/issues/648)
- 2.1 auto-extend by default, manual override but with warning / notification (https://github.com/ros/catkin/issues/649) (https://github.com/catkin/catkin_tools/pull/58#issuecomment-45556611)
- 2.1 report what was extended at the end of a build invocation (https://github.com/ros/catkin/issues/649) (https://github.com/catkin/catkin_tools/issues/63)
- 2.2 Better visibility / introspection, details need to be figured out
- 2.3 recommend to use “catkin build” once it is complete
- 2.4 keep CMAKE_PREFIX_PATH plus .catkin file but make it more visible / introspectable, consider switching in the future
- 2.5 ws command is a good idea, will be continued in the ticket (https://github.com/catkin/catkin_tools/issues/47)
- 2.6 Name for devel/install space: run / execute space? (https://github.com/ros-infrastructure/rep/issues/78)
    - setup space
    - fhs space
    - “invel” space (install+devel) (not really though)
- 2.7 list of entry points on catkin wiki page to point to the various different information, potentially also reference answers (https://github.com/ros/catkin/issues/650)
- 2.8 deprecation warning in cmi in Indigo once catkin build is available (https://github.com/catkin/catkin_tools/issues/55)
- 2.9 see 2.1
- 3.1 can be added (https://github.com/catkin/catkin_tools/issues/10 and https://github.com/catkin/catkin_tools/issues/27)
- 3.2 nice feature to be added (https://github.com/catkin/catkin_tools/issues/64)