



ROS BUNDLING

MIKE PURVIS



THE HISTORY

ros-install-osx, the evolution of development and deployment at Clearpath

THE PROBLEM

stable releases for industrial customers in a rolling release world

THE BUNDLE

the tools which power our solution, and some unanticipated benefits

THE FUTURE

how the tooling can further evolve to better to support this deployment model



HISTORY




Building ROS Indigo on Mac OS X, two years ago.

Installation Instructions for Indigo in OSX

This page describes how to install Indigo in OSX. OSX is not officially supported by ROS and the installation might fail for several reasons.

Contents

- 1. Installation instructions for Indigo in OSX
 - 1. Check
 - 1.1. Hardware
 - 1.2. Setup Environment
 - 1.3. Install Development Tools for Python
 - 1.4. Additional Tools
 - 2. Installation
 - 1. Building the catkin Packages
 - 1.1. Check for dependencies
 - 1.2. Building the catkin Workspace
 - 3. Installing a Custom Environment
 - 1. Update the workspace
 - 2. Manual package installation
 - 4. Troubleshooting
 - 1. Error logs
 - 1.1. Error logs
 - 1.2. Error logs
 - 2. Error logs to report
 - 1.1. Error logs to report from brew
 - 1.2. Error logs to report
 - 3. Error logs to report
 - 1.1. Error logs to report
 - 1.2. Error logs to report
 - 4. Error logs to report
 - 1.1. Error logs to report
 - 1.2. Error logs to report
 - 5. Error logs to report
 - 1.1. Error logs to report
 - 1.2. Error logs to report
 - 6. Error logs to report
 - 1.1. Error logs to report
 - 1.2. Error logs to report
 - 7. Error logs to report
 - 1.1. Error logs to report
 - 1.2. Error logs to report
 - 8. Error logs to report
 - 1.1. Error logs to report
 - 1.2. Error logs to report
 - 9. Error logs to report
 - 1.1. Error logs to report
 - 1.2. Error logs to report
 - 10. Error logs to report
 - 1.1. Error logs to report
 - 1.2. Error logs to report
 - 11. Error logs to report
 - 1.1. Error logs to report
 - 1.2. Error logs to report
 - 12. Error logs to report
 - 1.1. Error logs to report
 - 1.2. Error logs to report
 - 13. Error logs to report
 - 1.1. Error logs to report
 - 1.2. Error logs to report
 - 14. Error logs to report
 - 1.1. Error logs to report
 - 1.2. Error logs to report
 - 15. Error logs to report
 - 1.1. Error logs to report
 - 1.2. Error logs to report
 - 16. Error logs to report
 - 1.1. Error logs to report
 - 1.2. Error logs to report
 - 17. Error logs to report
 - 1.1. Error logs to report
 - 1.2. Error logs to report
 - 18. Error logs to report
 - 1.1. Error logs to report
 - 1.2. Error logs to report
 - 19. Error logs to report
 - 1.1. Error logs to report
 - 1.2. Error logs to report
 - 20. Error logs to report
 - 1.1. Error logs to report
 - 1.2. Error logs to report
 - 21. Error logs to report
 - 1.1. Error logs to report
 - 1.2. Error logs to report
 - 22. Error logs to report
 - 1.1. Error logs to report
 - 1.2. Error logs to report
 - 23. Error logs to report
 - 1.1. Error logs to report
 - 1.2. Error logs to report
 - 24. Error logs to report
 - 1.1. Error logs to report
 - 1.2. Error logs to report
 - 25. Error logs to report
 - 1.1. Error logs to report
 - 1.2. Error logs to report




The number of potential problems dwarfs the main body of the setup instructions.

Installation Instructions for Indigo in OSX

This page describes how to install Indigo in OSX. OSX is not officially supported by ROS and the installation might fail for several reasons.

Contents

- 1. Installation instructions for Indigo in OSX
 - 1. Check
 - 1.1. Hardware
 - 1.2. Setup Environment
 - 1.3. Install Development Tools for Python
 - 1.4. Additional Tools
 - 2. Installation
 - 1. Building the catkin Packages
 - 1.1. Check for dependencies
 - 1.2. Building the catkin Workspace
 - 3. Installing a Custom Environment
 - 1. Update the workspace
 - 2. Manual package installation
 - 4. Troubleshooting
 - 1. Error logs
 - 1.1. Error logs
 - 1.2. Error logs
 - 2. Error logs to report
 - 1.1. Error logs to report from brew
 - 1.2. Error logs to report
 - 3. Error logs to report
 - 1.1. Error logs to report
 - 1.2. Error logs to report
 - 4. Error logs to report
 - 1.1. Error logs to report
 - 1.2. Error logs to report
 - 5. Error logs to report
 - 1.1. Error logs to report
 - 1.2. Error logs to report
 - 6. Error logs to report
 - 1.1. Error logs to report
 - 1.2. Error logs to report
 - 7. Error logs to report
 - 1.1. Error logs to report
 - 1.2. Error logs to report
 - 8. Error logs to report
 - 1.1. Error logs to report
 - 1.2. Error logs to report
 - 9. Error logs to report
 - 1.1. Error logs to report
 - 1.2. Error logs to report
 - 10. Error logs to report
 - 1.1. Error logs to report
 - 1.2. Error logs to report
 - 11. Error logs to report
 - 1.1. Error logs to report
 - 1.2. Error logs to report
 - 12. Error logs to report
 - 1.1. Error logs to report
 - 1.2. Error logs to report
 - 13. Error logs to report
 - 1.1. Error logs to report
 - 1.2. Error logs to report
 - 14. Error logs to report
 - 1.1. Error logs to report
 - 1.2. Error logs to report
 - 15. Error logs to report
 - 1.1. Error logs to report
 - 1.2. Error logs to report
 - 16. Error logs to report
 - 1.1. Error logs to report
 - 1.2. Error logs to report
 - 17. Error logs to report
 - 1.1. Error logs to report
 - 1.2. Error logs to report
 - 18. Error logs to report
 - 1.1. Error logs to report
 - 1.2. Error logs to report
 - 19. Error logs to report
 - 1.1. Error logs to report
 - 1.2. Error logs to report
 - 20. Error logs to report
 - 1.1. Error logs to report
 - 1.2. Error logs to report
 - 21. Error logs to report
 - 1.1. Error logs to report
 - 1.2. Error logs to report
 - 22. Error logs to report
 - 1.1. Error logs to report
 - 1.2. Error logs to report
 - 23. Error logs to report
 - 1.1. Error logs to report
 - 1.2. Error logs to report
 - 24. Error logs to report
 - 1.1. Error logs to report
 - 1.2. Error logs to report
 - 25. Error logs to report
 - 1.1. Error logs to report
 - 1.2. Error logs to report



Actual installation instructions.

Stuff that might go wrong.

New approach used brewed python/numpy/scipy instead of the system versions of those packages, and built in parallel with catkin_tools, rather than catkin_make_isolated. Started life as a gist, eventually evolved into runnable script. Positive side effect has been a centralized bug-tracker and place to review and collaborate on fixes with a growing community of ROS Mac users.

<https://github.com/mikepurvis/ros-install-osx>


"This repo aims to maintain a usable, scripted, up-to-date installation procedure for ROS. The intent is that the install script may be executed on a bare Yosemite or El Capitan machine and produce a working desktop_full installation, including RQT, rviz, and Gazebo."


Initial Gist
Mar 2014

July 2015
Github Repo

June 2016
Brewed Python rosdeps

July 2016
OS X Travis Tests





```

rosinstall_generator desktop_full --rostdistro indigo
--deps --tar > desktop_full.rosinstall

wstool init -j8 src desktop_full.rosinstall

rosdep install --from-paths src --ignore-src --rostdistro indigo
--as-root pip:no -y

catkin config --install

catkin build

source install/setup.sh

```



Process is:

1. Generate a list of all packages which will be in the workspace.
2. Download the tarball for each package.
3. Install system dependencies using rosdep (stuff like CMake, Boost, PCL).
4. Giant parallel build!

```

[build] Found '225' packages in 0.0 seconds.
Starting >>> catkin
Finished <<< catkin [ 0.8 seconds ]
Starting >>> genmsg
Starting >>> angles
Starting >>> cmake_modules
Starting >>> convex_decomposition
Starting >>> cpp_common
Starting >>> angles
Starting >>> eigen_stl_containers
Starting >>> ivcon
Starting >>> convex_decomposition
Starting >>> media_export

```



This is what the build looks like in catkin_tools. We're not going to dwell here except to say that a catkin_tools is equivalent in approach to catkin_make_isolated, meaning that each package is configured as its own project, rather than being combined together. This is critical because plain CMake packages like orocos_kdl and catkin itself can't be combined into one workspace the way catkin packages can.

```

- tar:
  local-name: actionlib
  uri: https://github.com/ros-gbp/actionlib-release/archive/release/indigo/actionlib/1.11.6-0.tar.gz
  version: actionlib-release-release-indigo-actionlib-1.11.6-0

- tar:
  local-name: angles
  uri: https://github.com/ros-gbp/geometry_angles_utils-release/archive/release/indigo/angles/1.9.10-0.tar.gz
  version: geometry_angles_utils-release-release-indigo-angles-1.9.10-0

- tar:
  local-name: bond_core/bond
  uri: https://github.com/ros-gbp/bond_core-release/archive/release/indigo/bond/1.7.17-0.tar.gz
  version: bond_core-release-release-indigo-bond-1.7.17-0

...

```



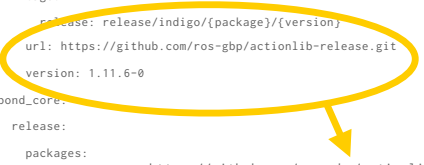

This is a snippet of what's inside the rosinstall file that wstool consumes. It's a list of tarball links which will be downloaded and unpacked to populate the source workspace.

```
actionlib:
  release:
    tags:
      release: release/indigo/{package}/{version}
      url: https://github.com/ros-gbp/actionlib-release.git
      version: 1.11.6-0
bond_core:
  release:
    packages:
      - bond
      - bond_core
      - bondcpp
      - bondpy
      - smclib
```



The `rosinstall_generator` tool uses as its source data the `distribution.yaml` which governs which version of every package is the current latest in a given distribution. So in this case, version 1.11.6-0 was the last bloomed release of `actionlib` into ROS Indigo. The build farm uses this data to generate Jenkins jobs to build debian packages, but `rosinstall_generator` uses it to generate source workspaces.

```
actionlib:
  release:
    tags:
      release: release/indigo/{package}/{version}
      url: https://github.com/ros-gbp/actionlib-release.git
      version: 1.11.6-0
bond_core:
  release:
    packages:
      - bond
      - bond_core
      - bondcpp
      - bondpy
      - smclib
```



The generator is capable of generating a list of git URLs, but downloading tarballs is much faster than cloning git repos, so there is logic to recognize known git hosts and supply tarball URLs with the `--tar` flag.

```
rosinstall_generator desktop_full --rostdistro indigo --deps --tar
```



This used to support github only, but we patched it to also recognize bitbucket and gitlab. Now the other missing piece here is the `--deps` flag. How does `rosinstall_generator` know which packages to pull in when you call for dependencies?

```
http://repositories.ros.org/rosdistro_cache/<distro>-cache.yaml.gz
<?xml version="1.0" encoding="utf-8"?><package format="2"><name>actionlib</name><version>0.5.1</version><description>Handles action server exceptions during loss of network</description><maintainer email="msafric@clearpathrobotics.com">msafric</maintainer><license>Clearpath Proprietary</license><buildtool_depend>catkin</buildtool_depend><build_depend>actionlib</build_depend><build_export_depend>actionlib</build_export_depend><exec_depend>roscpp</exec_depend><test_depend>roslint</test_depend></package>
<?xml version="1.0" encoding="utf-8"?><package name="actionlib" version="1.11.5"><description>The actionlib stack provides a standardized interface for interfacing with preemptable tasks. Examples of this include moving the base to a target location, performing a laser scan and returning the resulting point cloud, detecting the handle of a door, etc.</description><maintainer email="mikaels@osrfoundation.org">Mikael Arguedas</maintainer><license>BSD</license><url type="website">http://www.ros.org/wiki/actionlib</url><url type="bugtracker">https://github.com/ros/actionlib/issues</url><url type="repository">https://github.com/ros/actionlib</url><author>Eitanarder Eppstein</author><author>Vijay Pradeep</author><buildtool_depend>catkin</buildtool_depend><build_depend>roscpp</build_depend><build_export_depend>roscpp</build_export_depend><exec_depend>roscpp</exec_depend><test_depend>roslint</test_depend></package>
<?xml version="1.0" encoding="utf-8"?><package name="actionlib_msgs" version="1.11.9"><description>actionlib_msgs defines the common messages to interact with an action server and an action client. For full documentation of the actionlib API see the<a href="http://www.ros.org/wiki/actionlib">actionlib</a> package.</description><maintainer email="tfoote@osrfoundation.org">Tully Foote</maintainer><license>BSD</license><url type="website">http://www.ros.org/wiki/actionlib_msgs</url><author>Vijay Pradeep</author><buildtool_depend>catkin</buildtool_depend><build_depend>message_generation</build_depend><build_export_depend>message_generation</build_export_depend><exec_depend>message_runtime</exec_depend><test_depend>roslint</test_depend></package>
```

Every few minutes the buildfarm examines the distribution.yaml and updates a *secondary* yaml which is the contents of the distribution plus an additional dictionary which contains the complete package.xml string for every package. By accessing this cache, rosinstall_generator is able to quickly determine recursive dependencies.



So. That's ros-install-osx. The second piece of history is that Clearpath has executed a pivot in the past two years, transitioning from a research products company to focusing primarily on a full-stack solution for autonomous mobile robots in the industrial logistics space. This has meant a bunch of changes in how we deploy which are what have primarily driven the development of our bundle pipeline.

THE PROBLEMS

About a year ago, I officially joined the industrial team, as Firmware and Integration manager. Deployments to robots, developers, simulation environments were all based on bloomed deb packages as generated by buildbot-ros and then later ros_buildfarm. We had the following issues with this approach.



1

Rolling release. Robots and devs would all be running "testing debs" without any clear anchor on what versions of packages they actually had.



We had scripts which would use rospack list or rosversion to at least *report* the versions on a machine at a given time (for example, when a problem was observed, or a crash occurred), but knowing is only half the battle—it doesn't help you *reproduce* a bad state when trying to debug it later.

2

Coupled Repos. Having many interdependent packages in different repos meant having to bloom changes all the time to keep unit tests from breaking, rather than because there was a logical, releasable change.



Because ros_buildfarm sets up the test environment by installing the latest *released* version of all test dependencies.

The easy argument here is “don't have so many repos,” but this is not so simple, particularly when you consider our relationship to upstream. Take our controls code, which is spread across three repos: navigation, the planner, and the tracker, where navigation is a fork of upstream, and the planner and the tracker are proprietary. At this point our navigation fork is permanent and permanently private, but a year ago there was still a vague ambition to try to merge our

3

The release process was a debmirror snapshot of the testing repo, similar to how upstream periodically syncs packages to public.



Meaning that there was no sane way to put incremental bugfixes on a build — you were either on the bleeding edge or a stale snapshot. We experimented with injecting one-off bug fix debs into a release, but it was a very manual, error-prone process.

4

Meaningless package releases. For scale reasons, we strongly preferred to run the test fleet on binaries, so to get code into testing, developers would bloom broken stuff all the time.



Part of how I justify getting sucked into this project despite being the “firmware” manager is that we increasingly view the entire robot as an embedded target to be flashed, rather than a Linux PC on wheels with an ordinary computer lifecycle.

THE BUNDLE



“Lets centrally build a workspace of a bunch of our stuff, tar up the installspace, and use that on the robots.”



This was the proposal that my colleagues and I discussed as a starting point.

Based on ros-install-osx— just build everything! Why bother separating out an overlay?

all our stuff
"Lets centrally build a workspace of ~~a bunch of our stuff~~,
tar up the installspace, and use that on the robots."



We need versioning and a distribution scheme; let's just use a debs and an apt repo; those are already a thing.

all our stuff
"Lets centrally build a workspace of ~~a bunch of our stuff~~,
~~tar up~~ the installspace, and use that on the robots."
make a deb package of



So your first question might be "seriously, a deb package, in this age of sexy things like containers?"

all our stuff
"Lets centrally build a workspace of ~~a bunch of our stuff~~,
~~tar up~~ the installspace, and use that ~~on the robots~~ everywhere!
make a deb package of




```
virtualenv create foo
foo/bin/pip install -r requirements.txt
```

```
debootstrap trusty foo
chroot foo
```

```
docker build -t foo .
docker run -t -i foo /bin/bash
```

```
 vagrant box add ubuntu/trusty64
 vagrant up
 vagrant ssh
```



MORE ISOLATION

And it's true, isolation is a big thing right now. Type and degree of isolation varies.

```
virtualenv create foo
foo/bin/pip install -r requirements.txt
```

```
debootstrap trusty foo
chroot foo
```

```
docker build -t foo .
docker run -t -i foo /bin/bash
```

```
 vagrant box add ubuntu/trusty64
 vagrant up
 vagrant ssh
```



CATKIN

Base catkin is isolated in terms of filesystem only. Sort of a virtualenv for C++ CMake projects. We're not doing multiple apps, and port or CPU isolation would just get in our way, so we don't need a docker container— we can ship this thing as a deb, and then there's a lot of additional tools we can take advantage of in the debian ecosystem which don't exist for docker (and we don't have to run the daemon).

Parallel XZ Compression

<https://launchpad.net/~mikepurvis/+archive/ubuntu/dpkg>

When using dpkg-deb to compress a 2.2GB payload, this version of dpkg-deb takes, on a 12-core machine:

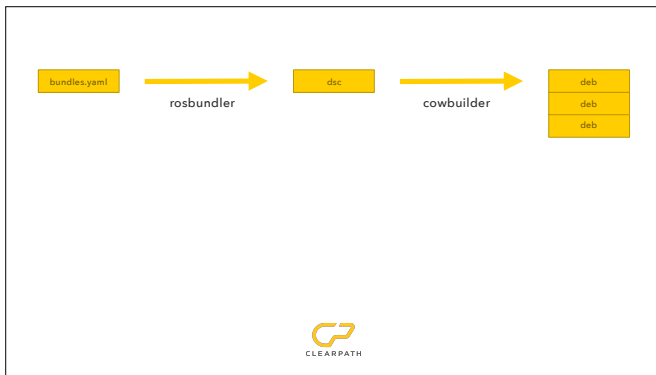
- 30 seconds to compress to 555MB (-x1)
- 42 seconds to compress to 521MB (-x3)
- 119 seconds to compress to 478MB (-x6)

Running the same test with dpkg 1.17.5 (trusty default):

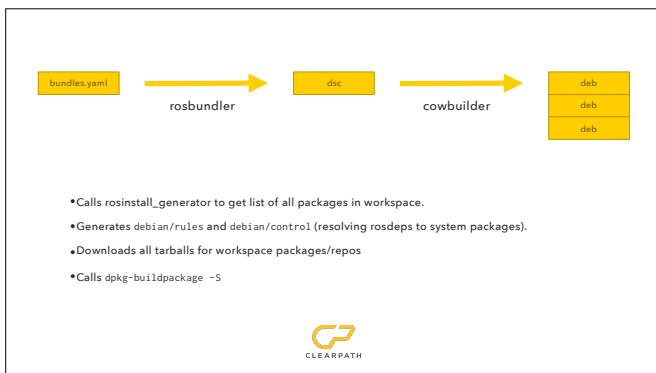
- 319 seconds to compress to 550MB (-x1)



If you're like me and you suddenly find yourself generating a bunch of large-ish debs, you'll quickly discover that the default compression for dpkg-deb is single-threaded xz, which can be extremely slow. Add this PPA to your 8 or 16 core buildslave and watch it scream through that end-of-build compression step.



So, here's basically our process. A bundle config file defines what our bundles are, where the rosdistro for each one is, and what packages should be used to "seed" rosinstall_generator.



Our rosbundler tool generates the required debian package metadata files, including instructions for how to build everything (the debian/rules file), and then downloads the tarballs, naming them according to debian conventions.

Keen observers will note that this doesn't actually address a lot of the problems listed above. By default, rosinstall_generator is always grabbing the latest released version of every package, so this bundle is equivalent to whatever the testing debs are at the moment that the bundle is built. How do we actually lock in the versions of the packages?

```
actionlib:
  release:
    tags:
      release: release/indigo/{package}/{version}
    url: https://github.com/ros-gbp/actionlib-release.git
    version: 1.11.6-0
bond_core:
  release:
    packages:
      - bond
      - bond_core
      - bondcpp
      - bondpy
      - smclib
```

ros/rosdistro/indigo/distribution.yaml


Well, here's the distribution file which actually defines the versions of everything. This lives in a git repo, and a git repo can be tagged.

```
actionlib:
  release:
    tags:
      release: release/indigo/{package}/{version}
      url: https://github.com/ros-gbp/actionlib-release.git
      version: 1.11.6-0
bond_core:
  release:
    packages:
      - bond
      - bond_core
      - bondcpp
      - bondpy
      - smclib
```

ros/rosdistro/indigo/distribution.yaml

to lock these versions

tag this repo




So we'll tag the repo with the number of our major release, and that'll forever give us a reference point to regenerate this configuration.

rosbundles/debian/changelog

```
clearpath-2.0 (2.0.2-0) trusty; urgency=medium
 * [redacted]
 -- Mike Purvis <mpurvis@clearpathrobotics.com> Thu, 23 Jun 2016 15:32:59 -0400

clearpath-2.0 (2.0.1-0) trusty; urgency=medium
 * [redacted]
 -- Mike Purvis <mpurvis@clearpathrobotics.com> Wed, 22 Jun 2016 16:04:05 -0400
```

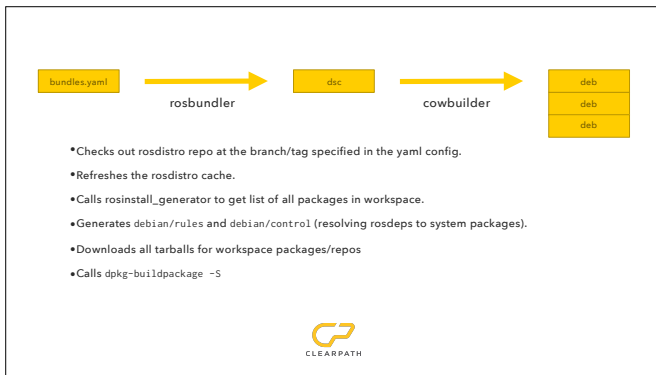


In fact, we use a regular Debian changelog file which lives alongside the bundles.yaml file, and the version specified in each changelog entry refers directly to a rosdistro tag.

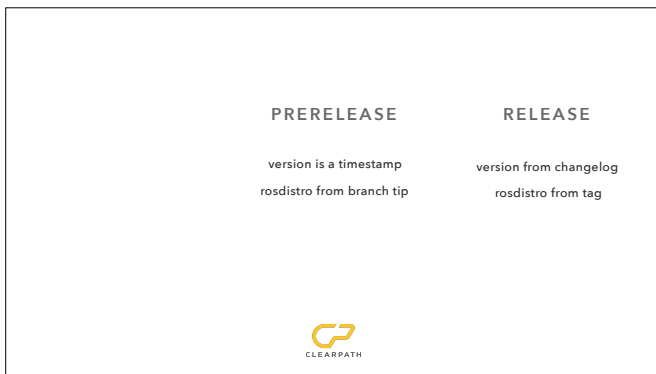
Each major release of our entire software stack is a tag on our rosdistro repo, locking the version of every package.



Big and bold so it can't be missed.



So to finish this out, when we're building a "release" bundle, we use the package versions from a tag of our rosdistro, rather than the master branch.



And prior to tagging a legit "release", we can still build a "prerelease", which is each package at its latest version. Eventually we may tag those as well— it would be nice to understand which repos have mutated when we're looking at a regression.



Now, we've talked a lot about the release stanza of the distribution.yaml entry— this is the entry whose git URL references a GBP repo, one managed by bloom.

```

actionlib:
  release:
    tags:
      release: release/indigo/{package}/{version}
      url: https://github.com/ros-gbp/actionlib-release.git
      version: 1.11.6-0
  source:
    type: git
    url: https://github.com/ros/actionlib.git
    version: indigo-devel

```



But the source stanza is actually where the bundle really comes into its full potential. Let's look at a new `roinstall_generator` flag.

```

roinstall_generator desktop_full --rostdistro indigo
--deps --tar --upstream-development

```



Adding `--upstream-development` means that instead of getting the source from the bloom GBP repo, we get it directly from the source repo.

roinstall_generator

```

- tar:
  local-name: actionlib
  uri: https://github.com/ros-gbp/actionlib-release/archive/release/indigo/actionlib/1.11.6-0.tar.gz
  version: actionlib-release-release-indigo-actionlib-1.11.6-0
- tar:
  local-name: actionlib
  uri: https://github.com/ros/actionlib/archive/indigo-devel.tar.gz
  version: actionlib-indigo-devel

```




Above is what we get by default— this is the latest released source, pulled from the GBP repo. Underneath is with the `--upstream-development` flag, this is the development branch from the upstream repo.

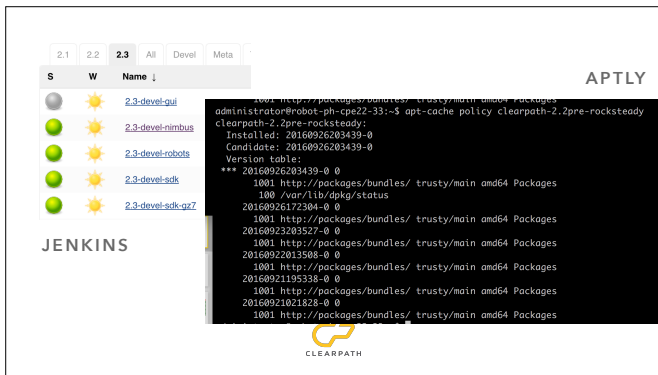
Can you see what's happening here? There's a new type of bundle that's possible, where we don't need to bloom at all.

DEVEL	PRERELEASE	RELEASE
version is a timestamp roscdistro from branch tip upstream devel sources	version is a timestamp roscdistro from branch tip	version from changelog roscdistro from tag
		

The devel bundle is built entirely of devel sources. Seems scary right? This has actually worked out phenomenally well for us, for a few reasons.

DEVEL	<ul style="list-style-type: none">• test fleet is always running the latest devel code.• when something breaks upstream (eg, ros_comm), we catch it immediately.• we're motivated to push patches, since we get the fix right away.• we bloom for logical functional releases releases only.• devel bundle can feed other testing workflows, including multi-robot simulation testing and unit test jobs.
version is a timestamp roscdistro from branch tip upstream devel sources	
	

MAJOR VERSION	BUNDLE TYPE		
	devel	prerelease	release
	2.0	clearpath-2.0devel-robots clearpath-2.0devel-nimbus clearpath-2.0devel-sdk	clearpath-2.0pre-robots clearpath-2.0pre-nimbus clearpath-2.0pre-sdk
	2.1	clearpath-2.1devel-robots clearpath-2.1devel-nimbus clearpath-2.1devel-sdk	clearpath-2.1pre-robots clearpath-2.1pre-nimbus clearpath-2.1pre-sdk
	2.2	clearpath-2.2devel-robots clearpath-2.2devel-nimbus clearpath-2.2devel-sdk	clearpath-2.2pre-robots clearpath-2.2pre-nimbus clearpath-2.2pre-sdk
			



In terms of our actual infrastructure, we build the bundles on Jenkins, and then send them to Aptly. Aptly is super cool for many reasons— two of them are that it can run as a REST service, so you don't have to deal with SSH or SCP, you can just POST packages to it with curl if you want.

► actionlib	Mikael A	1.11.5-1	1.11.6-0	indigo-devel	2	maintained
► angles	Ioan S	1.9.10-0	1.9.10-0	master	7	
► bfl	Wim M	0.7.0-6	0.7.0-6	release/indigo	6	maintained
► bond_core	Mikael A	1.7.17-0	1.7.17-0	master	2	maintained
► catkin	Dirk T	0.6.18-0	0.6.18-0	indigo-devel	3	maintained
► class_loader	Mikael A	0.3.4-0	0.3.4-0	indigo-devel	3	maintained
► cmake_modules	William W	0.3.3-0	0.3.3-0	0.3-devel	✓	maintained
► common_msgs	Tully F	1.11.9-0	1.11.9-0	indigo-devel	4	maintained
► control_msgs	Adolfo RT	1.3.1-0	1.3.1-0	indigo-devel	2	maintained
► control_toolbox	Sachin C	1.13.2-0	1.13.2-0	indigo-devel	10	maintained
► diagnostics	Brice R	1.8.10-0	1.8.10-0	indigo-devel	✓	maintained
► driver_common	Chad R	1.6.8-2	1.6.8-2	indigo-devel	2	Will be released only as long as
► dynamic_reconfigure	Mikael A	1.5.44-0	1.5.44-0	master	2	maintained
► eigen_stl_containers	Ioan S	0.1.4-0	0.1.4-0	master	7	
► filters	Tully F	1.7.4-0	1.7.4-0	hydro-devel	4	maintained
► gazebo_ros_pkgs	John H	2.4.11-0	2.4.11-0	indigo-devel	✓	developed
► genecpp	Dirk T	0.5.5-0	0.5.5-0	indigo-devel	✓	maintained
► gentliap	Dirk T	0.4.15-0	0.4.15-0	groovy-devel	8	maintained

This is our build status page, which is source repo oriented as opposed to the build farm one, which is deb package oriented.

► actionlib	Mikael A	1.11.5-1	1.11.6-0	indigo-devel	2	maintained
► angles	Ioan S	1.9.10-0	1.9.10-0	master	7	
► bfl	Wim M	0.7.0-6	0.7.0-6	release/indigo	6	maintained
► bond_core	Mikael A	1.7.17-0	1.7.17-0	master	2	maintained
▼ catkin	Dirk T	0.6.18-0	0.6.18-0	indigo-devel	3	maintained
<div> <div> <div>catkin</div> <div> Maintainers: Dirk Thomas Dependencies: cmake, python-argparse, python-catkin-pkg, python-empy, python-mock, python-nose Active Branches (1): kinetic-devel </div> </div> <div> <div>Changelog</div> <div> <pre> ***** Changelog for package catkin ***** 0.6.18 (2016-03-18) ----- * expose format 2 style dependencies as CMake variables ("#787") 0.6.17 (2016-03-05) </pre> </div> <div> <div>New Commit Messages</div> <div> <pre> - Improve doc about catkin_package - Merge pull request #791 from Nikol doc: fix format 2 howto to suggest t message_runtime - 0.6.18 </pre> </div> </div> </div> </div>						
► class_loader	Mikael A	0.3.4-0	0.3.4-0	indigo-devel	3	maintained
► cmake_modules	William W	0.3.3-0	0.3.3-0	0.3-devel	✓	maintained
► common_msgs	Tully F	1.11.9-0	1.11.9-0	indigo-devel	4	maintained

Each row pops open to reveal the changelog and commits which have occurred between the last release and the current level branch.

```
$ rosdistro_freeze_source -h
Freeze a rosdistro's source branch versions to hashes or tags. If
neither
--release-version nor --release-tag are specified, the hashes of the
current
devel branches are used.
```

positional arguments:

index	Path to a local index.yaml file.
dist_names	The names of the distributions (default: all)

optional arguments:

-h, --help	show this help message and exit
-j JOBS, --jobs JOBS	How many worker threads to use.
-q, --quiet	Suppress updating status bar (for script/CI usage).
--release-version	Freeze to the hash of current release tag.
--release-tag	Freeze to name of current release tag.

rostdistro_freeze_source

```
actionlib:
source:
type: git
url: https://github.com/ros/actionlib.git
version: indigo-devel
```



rostdistro_freeze_source

```
actionlib:
source:
type: git
url: https://github.com/ros/actionlib.git
version: indigo-devel
```


```
actionlib:
source:
type: git
url: https://github.com/ros/actionlib.git
version: 32ce9fdb3194228795a2f31faee8259dd57ae452
```

```
actionlib:
source:
type: git
url: https://github.com/ros/actionlib.git
version: 1.11.6
```



Once a major release for us goes into maintenance, we “freeze” the devel branches to point to the tag of the last release. This allows development to move forward, while still giving us a stable level build upon which to put hot fixes.

	Maintainer	2.1.0	2.1.1	2.1.2	2.1.3	2.1.4	2.1.5	latest	2.2	devel
Clearpath Proprietary										
▶ appliance_status	Gagan S	0.0.3-0	0.0.3-0	0.0.3-0	0.0.3-0	0.0.3-0	0.0.3-0	0.0.3-0	0.0.6-0	0.0.3 ✓
▶	Enrique F	0.0.6-0	0.0.6-0	0.0.6-0	0.0.6-0	0.0.6-0	0.0.6-0	0.0.6-0	0.0.7-0	0.0.8 ✓
▶	Enrique F	0.0.5-0	0.0.5-0	0.0.5-0	0.0.5-0	0.0.5-0	0.0.5-0	0.0.5-0	0.0.6-0	0.0.5 ✓
▶	Enrique F	0.0.3-0	0.0.3-0	0.0.3-0	0.0.3-0	0.0.3-0	0.0.3-0	0.0.3-0	0.0.4-0	0.0.3 ✓
▶	Enrique F	0.0.1-0	0.0.1-0	0.0.1-0	0.0.1-0	0.0.1-0	0.0.1-0	0.0.1-0	0.0.2-0	0.0.1 ✓
▶ atlas	Andrew B	2.1.13-0	2.1.13-0	2.1.14-0	2.1.14-0	2.1.17-0	2.1.18-0	2.1.18-0	2.2.3-0	devel:2.1 ✓
▶ audio_indication	Mike P	0.3.0-0	0.3.0-0	0.3.0-0	0.3.0-0	0.3.0-0	0.3.0-0	0.3.0-0	0.5.0-0	0.3.0 ✓
▶	Alex B	0.2.0-0	0.2.0-0	0.2.0-0	0.2.0-0	0.2.0-0	0.2.0-0	0.2.0-0	0.5.0-0	0.2.0 ✓
▶ clearpath_gazebo_worlds	Jordan L	0.3.4-0	0.3.4-0	0.3.4-0	0.3.4-0	0.3.4-0	0.3.4-0	0.3.4-0	0.5.2-0	0.3.4 ✓
▶ cmd_vel_rate_monitor	Steph M	0.3.1-0	0.3.1-0	0.3.1-0	0.3.1-0	0.3.1-0	0.3.1-0	0.3.1-0	0.5.1-0	0.3.1 ✓
▶ control_selection	Alex B	0.4.0-0	0.4.0-0	0.4.0-0	0.4.0-0	0.4.0-0	0.4.0-0	0.4.0-0	0.5.0-0	0.4.0 ✓


CLEARPATH

DEMONSTRATION

<https://github.com/mikepurvis/ros-bundling>



catkin test, catkin docs.
dropping release stanzas and blooming, consi.
once the ubuntu store is open source, investigate packaging as a snap.

THE FUTURE



catkin docs

- implement as catkin_tools plugin
- make rosdod_lite into plain python package, so the plugin could depend on it.
- allow building all docs in one big workspace, using the same parallel executor model as catkin build.

https://github.com/catkin/catkin_tools/issues/381



One of the big wins with this approach would be potentially giving more opportunities to supply global overrides. So, same as how building the bundle lets you do fun things like set compiler flags that correspond to the architecture of your robot PC, or archive debug symbols, a future catkin docs might allow injecting a Doxygen theme, etc.

catkin test

- implement as catkin_tools plugin
- safe parallel testing of a workspace which may contain cmake and catkin (and ament?) packages.
- have a scheme for tests to declare and then properly mutex shared testing resources, like the Gazebo port.
- would work for CI, but also be a better story locally for devs using catkin_tools.

https://github.com/catkin/catkin_tools/issues/397



One of the other major functional gaps in catkin_tools right now is not having a good story for running tests.

source only

- drop release stanzas from internal distribution yaml.
- only bloom packages which are going for public release, otherwise do only catkin_prepare_release to uprev package.xml and create tag.
- depends on source manifest caching for dependency resolution.



source only

actionlib:

source:

type: git

url: <https://github.com/ros/actionlib.git>

version: indigo-devel

release_version: 1.11.6

- drop release stanzas from internal distribution yaml.
- only bloom packages which are going for public release, otherwise do only catkin_prepare_release to uprev package.xml and create tag.



Snaps

- Packaging as a squashfs is compelling.
- Delta updates are compelling.
- Multiple release streams (edge, RC, etc) is very compelling.
- Blocked on the repository code being open sourced.



THANKS

William and Dirk

Jon Bohren

Nikolaus Demmel

Mike Ferguson

Contributors to ros-install-osx

Clearpath



<https://ottomotors.com/company/careers>

OSRF staff for maintaining the tools and entertaining my crazy ideas and PRs.
Jon for re-architecting the catkin_tools backend.

Nikolaus for helping with OS X issues, and contributing the no-sudo option to
roscdep.

Mike Ferguson for buildbot-ros and getting me into debian packaging and
cowbuilder.

Clearpath for being an awesome place to work, and for allowing me time to
work on this, and especially my colleagues on the platform and tools/
infrastructure teams. Obviously, we are hiring.