## WORKSPACES

### Create Workspace

```
mkdir catkin_ws && cd catkin_ws
wstool init src
catkin_make
source devel/setup.bash
```

### Add Repo to Workspace

```
roscd; cd ../src
wstool set repo_name --git http://github.com/org/repo_name.git --version=hydro-devel
wstool up
```

### Resolve Dependencies in Workspace

```
sudo rosdep init  # only once
rosdep update
rosdep install --from-paths src --ignore-src --rosdistro=hydro -y
```

## PACKAGES

### Create a Package

```
catkin_create_pkg package_name [dependencies ...]
```

### Package Folders

- **include/package_name**: C++ header files
- **src**: Source files.
- **scripts**: Python nodes and scripts
- **msg, srv, action**: Message, Service, and Action definitions

### Release Repo Packages

```
catkin_generate_changelog
# review & commit changelogs
catkin_prepare_release
bloom-release --track hydro --ros-distro hydro repo_name
```

### Reminders

- Testable logic
- Publish diagnostics
- Desktop dependencies in a separate package

## CMakeLists.txt

### Skeleton

```
cmake_minimum_required(VERSION 2.8.3)
project(package_name)
find_package(catkin REQUIRED)
catkin_package()
```

### Package Dependencies

To use headers or libraries in a package, or to use a package's exported CMake macros, express a build-time dependency:

```
find_package(catkin REQUIRED COMPONENTS roscpp)
```

Tell dependent packages what headers or libraries to pull in when your package is declared as a catkin component:

```
catkin_package(
    INCLUDE_DIRS include
    LIBRARIES ${PROJECT_NAME}
    CATKIN_DEPENDS roscpp)
```

Note that any packages listed as CATKIN_DEPENDS dependencies must also be declared as a `<run_depend>` in `package.xml`.

### Messages, Services

These go after `find_package()`, but before `catkin_package()`.

Example:

```
find_package(catkin REQUIRED COMPONENTS message_generation std_msgs)
add_message_files(FILES MyMessage.msg)
add_service_files(FILES MyService.srv)
generate_messages(DEPENDENCIES std_msgs)
catkin_package(CATKIN_DEPENDS message_runtime std_msgs)
```

## Build Libraries, Executables

Goes after the `catkin_package()` call.

```
add_library(${PROJECT_NAME} src/main)
add_executable(${PROJECT_NAME}_node src/main)
target_link_libraries(
    ${PROJECT_NAME}_node ${catkin_LIBRARIES})
```

## Installation

```
install(TARGETS ${PROJECT_NAME}
    DESTINATION ${CATKIN_PACKAGE_LIB_DESTINATION})
install(TARGETS ${PROJECT_NAME}_node
    DESTINATION ${CATKIN_PACKAGE_BIN_DESTINATION})
install(PROGRAMS scripts/myscript
    DESTINATION ${CATKIN_PACKAGE_BIN_DESTINATION})
install(DIRECTORY launch
    DESTINATION ${CATKIN_PACKAGE_SHARE_DESTINATION})
```

## RUNNING SYSTEM

### Run ROS using plain:

```
roscore
```

Alternatively, roslaunch will run its own roscore automatically if it can’t find one:

```
roslaunch my_package package_launchfile.launch
```

Suppress this behaviour with the `--wait` flag.

### Nodes, Topics, Messages

```
rosnode list
rostopic list
rostopic echo cmd_vel
rostopic hz cmd_vel
rostopic info cmd_vel
rosservice show geometry_msgs/Twist
```

### Remote Connection

Master's ROS environment:

- `ROS_IP` or `ROS_HOSTNAME` set to this machine's network address.
- `ROS_MASTER_URI` set to URI containing that IP or hostname.

Your environment:

- `ROS_IP` or `ROS_HOSTNAME` set to your machine's network address.
- `ROS_MASTER_URI` set to the URI from the master.

To debug, check ping from each side to the other, run roswtf on each side.

### ROS Console

Adjust using `rqt_logger_level` and monitor via `rqt_console`. To enable debug output across sessions, edit the `$HOME/.ros/config/rosconsole.config` and add a line for your package:

```
log4j.logger.ros.package_name=DEBUG
```

And then add the following to your session:

```
export ROSCONSOLE_CONFIG_FILE=$HOME/.ros/config/rosconsole.config
```

Use the roslaunch `--screen` flag to force all node output to the screen, as if each declared `<node>` had the output="`screen`" attribute.

## www.clearpathrobotics.com/ros-cheat-sheet

© 2014 Clearpath Robotics, Inc. All Rights Reserved.