AARNET.org sources
deb http://mirror.aarnet.edu.au/pub/ubuntu/archive/ raring main
deb-src http://mirror.aarnet.edu.au/pub/ubuntu/archive/ raring main

ROS

Installing

sudo apt-get install ros-hydro-desktop-full	Desktop Full

If you installed ROS from a package manager like apt, then those packages will not be write accessible and should not be edited by you the user.
When working with ROS packages from source or when creating a new ROS package, you should always work in a directory that you have access to, like your home folder.

ROS GENERAL

topic

topic	data source/sink to which messages are posted to or subscribed from. a topic consists of a namespace/topicName.
source <space>/setup.bash	lets you find your project using "rospack find" removes references to other projects (ie cancels previous “source <space>/setup.bash”) sets ROS_PACKAGE_PATH according to setup.bash. <space> can be 'devel', 'install' or a fully qualified path to setup.bash

node

node	an executable that is connected to the ROS network

Naming Convention

Package Names	all alpha must be lowercase must start with a letter can include numbers & underscores
Variable Names	must start with a letter alpha can be upper and lower case can include numbers & underscores

ROS PACKAGES

Creating a new Workspace

- "mkdir -p ~/ros/projects/<projectName>/src"	create folder (top level source)
- "cd ~/ros/projects/<projectName>/src"	navigate to it.
- "catkin_init_workspace"	initialise the workspace (creates CMakeList.txt)
- "cd .."	navigate up 1 level
- "catkin_make"	builds the empty project & creates ./build & ./devel
- "source devel/setup.bash"	

Returning to a Workspace

- "cd ~/ros/projects/<projectName>"	navigate to it.
- "source devel/setup.bash"	lets you find your project using "rospack find" removes references to other projects (ie cancels previous “source devel/setup.bash”)

Creating a catkin package

- "cd ~/ros/projects/<workspace>/src"	navigate to "src" folder of the workspace.
- "catkin_create_pkg <packageName> [depend1] [depend2] …"	creates the <packageName> folder under “src” and creates **CmakeLists.txt** and **package.xml** in that folder. Package names start with a lower case letter and can only contain lower case letter, digits, and underscores.

Customise the package.xml file

Edit the package.xml file and fill out all the fields:

- description
- maintainer name/email
- author name/email
- license
- dependencies
    - build_depend
    - buildtool_depend
    - run_depend
    - test_depend

Build packages with catkin_make

- "cd <workspaceRootFolder>"
- catkin_make

any packages in "src" are built to the "build" folder
only **generated** files will appear in the "build" folder
setup.*sh generated files also go in the "build" folder

ROS COMMANDS

rqt

rqt	runs the GUI framework from which you can select various tools (see ROS TOOLS below for <u>some</u> of them) (Available in 13.04 as a package)

rospack

rospack find <packageName>	finds the path of the package name. If "source devel/setup.bash" wasn't run, it won't find the workspace but will find it in the install location (/opt/ros) if it exists there.
rospack depends1 <packageName>	lists the 1st order (primary) dependancies.
rospack depends <packageName>	lists the package's dependancies recursively.
rospack depends1 <dependency>	lists the dependancies of the dependent library, which is itself a package.

rosstack

rosstrack -h	provides information about stacks similar to **rospack**

roscd

roscd <packageName>	changes to the folder where the package is located. Uses the ROS_PACKAGE_PATH variable to search.
roscd log	changes folder to ROS' log file folder.

rosls

rosls <packageName[/subdir]>	similar to linux command 'ls'

roscp

roscp <packageName> <fileToCoptPath> <copyPath> eg: roscp rospy_tutorials AddTwoInts.srv srv/AddTwoInts.srv	copies files

catkin_make

catkin_make <make_target>	builds the make_target located in the “src” folder.
catkin_make install	also generates an install target in the “install” folder
catkin_make install -DCMAKE_INSTALL_PREFIX=/opt/ros/hydro	sets the install target of the package.
catkin_make install -DCATKIN_DEVEL_PREFIX=/home/nap/...	sets target for the development output of the make.
catkin_make --source my_src	allows to use a source location other than the standard “src” folder.

roscore

roscore	runs the core service that provides the communication

rosnode

rosnode list	lists the names of all the nodes
rosnode info /<node>	lists the Publications, Subscriptions, Services the node uses. Also shows info concerning 'roscore'
rosnode cleanup	cleans up the node list. when nodes have been closed using **ctrl-c** in the terminal instead of closing the window, the sometimes linger in the list **OR** $ROS_HOSTNAME environment variable has not been defined as per <u>N/W Setup - Sgl M/C Configuration</u>
rosnode ping <nodeName>	similar to 'ping' but using the rosnode. (Note that the name of the node can be customised using a remapping argument (see below).

rosrun

rosrun <packageName> <node_name>	runs the node
rosrun <packageName> <node_name> <remappingArgument>	specifying **<remappingArgument>** allows the initial parameters of the node to be set (eg: *rosrun turtlesim turtlesim_node __name:=Fighto*) **__name:=** changes the name of the node displayed in the node list.

rostopic

rostopic list [<topic>] [option flags] lists the active topics

-v	verbose
-p	publishers only
-s	subscribers only
-b <filename>	list topics in .bag file
--bag <filename>	list topics in .bag file

rostopic bw <topic> displays bandwidth used by this topic

eg: *rostopic bw turtle1/cmd_vel*

```
average: 3.00KB/s:
mean: 0.05KB min: 0.05KB max: 0.05KB window: 100
```

rostopic hz <topic> echos the publishing rate of the topic

```
average rate: 62.091
min: 0.013s max: 0.032s std dev: 0.00162s window: 121
```

rostopic echo <topic> echos the contents of the topic's messages

rostopic type <topic> echos the data types of the message

rostopic type rosout and
rostopic type rosout_agg are of rosgraph_msgs/Log type.

Note that these do not use the <namespace/topicName> input format.
The output represents the class and type

rostopic pub <topic> <msg_type> [args] manually publish data to the topic.

eg rostopic pub -1 /turtle1/cmd_vel geometry_msgs/Twist -- '[2.0, 0.0, 0.0]' '[0.0, 0.0, 1.8]'

-1	publish the message once only
/turtle1/cmd_vel	topic to publish to (turtle1 is the generic name of the node, if the node wasn't 'spawned') (use the correct name to differentiate between similar 'spawned' nodes)
geometry_msgs/Twist	message type
--	indicates that parameters following are ALL required
'[]' '[]'	the parameters being passed
-r 1	not when **-1** is used frequency in Hz for the message to be published.

rosmsg

rosmsg [command] -h	display help (including sub-commands)
rosmsg show <topic>	displays the data structure of the type
rosmsg show <[packageName/]messageType>	displays the data structure of the type

rosmsg md5

rosmsg package

rosmsg packages

rosservice

rosservice list — displays a list of services the active nodes provide

eg:

```
/Fighto/get_loggers
/Fighto/set_logger_level
/clear
/kill
/reset
/rosout/get_loggers
/rosout/set_logger_level
/rostopic_3389_1387282241399/get_loggers
/rostopic_3389_1387282241399/set_logger_level
/spawn
/turtle1/set_pen
/turtle1/teleport_absolute
```

/turtle1/teleport_relative

rosservice type <serviceName>	displays the type of the argument associated with the service **std_srvs/Empty** means no parameters passed or received eg: spawn uses turtlesim/Spawn
rosservice call <serviceName> [args]	invokes the **serviceName** requested with the **args** provided
rosservice find <serviceType>	displays all services that use this type eg: std_srvs/Empty → /reset & /clear
rosservice uri <serviceName>	displays the port used by this service eg: clear uses rosrpc://NBA:43578 (for this session)

rosparam

rosparam list	displays a list of rosparams
rosparam set <paramName> [args]	sets paramName to args sometimes need to followup with a rosservice call
rosparam get [nameSpace/]<paramName>	displays the value of paramName **rosparam get /** returns all parameter values with a nameSpace, displays value in that nameSpace
rosparam dump <fileName> [paramName]	dump parameter values to fileName (with name qualifiers) with a paramName, dumps the value of that parameter (without name qualifier)
rosparam load <fileName> [nameSpace]	load the contents of filename with a nameSpace, values are loaded into the nameSpace
rosparam delete	

rossrv

rossrv show <serviceType>	displays the data structure of the serviceType (similar to rostopic type)
rossrv show [packageName/]<serviceType>	displays the data structure of the serviceType

roslaunch

roslaunch <packageName> <filename.launch>	in the packageName folder (workspace/src/packageName) **mkdir launch** and **cd** into it. See: turtlemimic.launch

rosed

rosed <packageName> <fileName>	enables editing of file within the package uses extention, so if not supplied, will open the executable version, not the source (TAB completion works here too)

rosbag

rosbag record -a [-O <fileName>]	records all (-a) publised messages to a file in current folder CTRL-C to finish recording
rosbag record [-O <fileName>] [topic]...	only records messages to listed topics in filename CTRL-C to finish recording
rosbag info <fileName>	summarises the contents of the bagfile

rosbag play [-d <waitTime>] [-s <startTime>] [-r <scaleFactor>] <fileName>

kill any nodes that are generating messages (eg turtle_teleop_key)

waitTime	= time to wait before commencing replay
startTime	= time stamp from which to replay
scaleFactor	= accelerate the replay (x>1) or decelerate the replay (1>x>0)

*** rosbag record is not an exact replication.

The reason for this is that the path tracked by turtlesim is very sensitive to small changes in timing in the system, and rosbag is limited in its ability to exactly duplicate the behaviour of a running system in terms of when messages are recorded and processed by rosrecord, and when messages are produced and processed when using rosplay. For nodes like turtlesim, where minor timing changes in when command messages are processed can subtly alter behavior, the user should not expect perfectly mimicked behaviour.

ROS Tools

rqt_graph

rosrun rqt_graph rqt_graph	shows the 'graph' of the interaction between nodes. See: rqt_graph_pub.png

rqt_plot

rosrun rqt_plot rqt_plot	shows a plot of the data for a topic See: rqt_plot.png

rqt_consol

rosrun rqt_consol rqt_consol	opens a consol that displays output from nodes See: target=rqt_console(turtlesimstart).png

rqt_logger_level

rosrun rqt_logger_level rqt_logger_level	opens a consol that enables user to select nodes and set the verbosity (Debug/Info/Warn/Error/Fatal) level

roswtf

roswtf	analyses the ROS install and reports on settings, warnings, and errors

General ROS related commands

export | grep ROS printenv | grep ROS	Display Enviroment Variables. These two performe the same operation but the display is different.
echo $ROS_PACKAGE_PATH	display a particular environment variable
rostopic type /turtle1/cmd_vel | rosmsg show	combining the output of rostopic type with rosmsg show (standard linux piping)
rosrun turtlesim turtle_teleop_key	keyboard input. NOT teleop_turtle_key (for some reason)
export ROS_PACKAGE_PATH=~/<distro>_workspace/sandbox:$ROS_PACKAGE_PATH	manually adding to the ROS_PACKAGE_PATH
export EDITOR='nano'	in .bashrc, makes 'nano' the default ROS editor could be any editor eg: vim, emacs, gedit

Basic ROS Package Structure (using my root ROS package folder)

before a BUILD:

```
/ros/projects/workspace_folder/                         -- WORKSPACE
        src/                                            -- SOURCE SPACE
              CMakeLists.txt                            -- 'Toplevel' CMake file, provided by catkin
              package_name/                             -- This is the package name, in this case it's "package_name"
                     CMakeLists.txt                     -- CMakeLists.txt file for package_name
                     package.xml                        -- Package manifest for package_name
                     src/                               -- This is where your source files are
                     msg/                               -- This is where message definitions go (if you have such)
                     srv/                               -- This is where service definitions go (if you have such)
                     ...                                -- You will see 'cmake' and 'include' in some packages.
              another_package/                          -- This is the package name, in this case it's "anoother_package"
                     CMakeLists.txt                     -- CMakeLists.txt file for this package
                     package.xml                        -- Package manifest for this package
                     src/                               -- This is where your source files are
                     ...                                -- And other folders as required
              …
```

Notes:
Naming of packages is explained above near the start of this document, and in these file structure diagrams, numbers are used in package names only indicate multiples.

The "After BUILD" file structure is incomplete as at the time of writing. The 'devel/' folder also includes shared libraries that have been built.

after a BUILD:

```
/ros/projects/workspace_folder/                              -- WORKSPACE
    src/                                                     -- SOURCE SPACE
        CMakeLists.txt                                       -- 'Toplevel' CMake file, provided by catkin
        package_1/
            CMakeLists.txt                                   -- CMakeLists.txt file for package_1
            package.xml                                      -- Package manifest for package_1
            CMakeLists.txt                                   -- CMakeLists.txt file for package_name
            package.xml                                      -- Package manifest for package_name
            src/                                             -- This is where your source files are
            msg/                                             -- This is where message definitions go (if you have such)
            srv/                                             -- This is where service definitions go (if you have such)
            ...                                              -- You will see ‘cmake’ and ‘include’ in some packages.
        package_n/
            CMakeLists.txt                                   -- CMakeLists.txt file for package_n
            package.xml                                      -- Package manifest for package_n
            src/                                             -- This is where your source files are
            ...                                              -- And other folders as required
        ...
    build/
    devel/
        etc/
            catkin/
                profile.d/
                    05.catkin_make.bash
                    05.catkin_make_isolated.bash
                    05.catkin-test-results.sh
        include/
            turtlesim/                                       -- C++ Files
                Color.h                                      -- C++ Files
                Kill.h                                       -- C++ Files
                …                                            -- C++ Files
        lib/
            pkgconfig/
                turtlesim.pc
            python2.7/                                       -- Python Files
                dist-packages/                               -- Python Files
                    turtlesim/                               -- Python Files
                        msg/                                 -- Python Files
                            _Color.py                        -- Python Files
                            _Color.pyc                       -- Python Files
                            __init__.py                      -- Python Files
                            __init__.pyc                     -- Python Files
                            …                                -- Python Files
                        srv/                                 -- Python Files
                            __init__.py                      -- Python Files
                            __init__.pyc                     -- Python Files
                            _Kill.py                         -- Python Files
                            _Kill.pyc                        -- Python Files
                            …                                -- Python Files
                        __init__.py                          -- Python Files
                        __init__.pyc                         -- Python Files
            turtlesim/
                draw_square                                  -- executable in turtlesim
                mimic                                        -- executable in turtlesim
                turtlesim_node                               -- executable in turtlesim
                turtle_teleop_key                            -- executable in turtlesim
        share/
            common-lisp/                                     -- Lisp Files
                ros/                                         -- Lisp Files
                    turtlesim/                               -- Lisp Files
                        msg/                                 -- Lisp Files
                            Color.lisp                       -- Lisp Files
                            _package.lisp                    -- Lisp Files
                            _package_Color.lisp              -- Lisp Files
                            …                                -- Lisp Files
                            turtlesim-msg.asd                -- Lisp Files
                        srv/                                 -- Lisp Files
                            Kill.lisp                        -- Lisp Files
                            _package.lisp                    -- Lisp Files
                            _package_Kill.lisp               -- Lisp Files
                            …                                -- Lisp Files
                            turtlesim-srv.asd                -- Lisp Files
            turtlesim/
                cmake/
                    turtlesimConfig.cmake
                    turtlesimConfig-version.cmake
                    turtlesim-msg-extras.cmake
                    turlesim-msg-paths.cmake
        env.sh
        setup.bash
setup.sh
setup.zsh
setup_util.py
.catkin
.rosinstall
```

CODING for ROS

MSG

msg file	simple text files that describe the fields of a ROS message. They are used to generate source code for messages in different languages.

SRV

srv file	describes a service. It is composed of two parts: a request and a response. (ie Client/Server)

Types available for MSG & SRV files:

See: http://wiki.ros.org/msg for more details

- bool
- int8, int16, int32, int64 (plus uint*)
- float32, float64
- string (utf-8 only)
- time, duration
- other msg files
- variable-length array[] and fixed-length array[C]

Example MSG file structure:

```
Header header
string child_frame_id
geometry_msgs/PoseWithCovariance pose
geometry_msgs/TwistWithCovariance twist
```

Header header: contains timestamp & coordinate frame

SRV files are like MSG but have two sections separated by '---'

Example of a SRV file structure:

```
int64 A
int64 B
---
int64 Sum
```

Using MSG

in ./msg
create the msg file
confirm that <build_depend>message_generation</build_depend> and <run_depend>message_runtime</run_depend> are in the package.xml file
add/ensure that 'message_generation' is part of the find_package(catkin REQUIRED COMPONENTS ...) call in the CMakeLists.txt file (usually at the start of the file). (Note: *find_package(catkin REQUIRED COMPONENTS ...) cascades through the build. But it is risky not to include it in each package's CmakeLists.txt file*.)
add/ensure that 'message_runtime' is part of the catkin_package(CATKIN_DEPENDS ...) call in CmakeLists.txt
In CMakeList.txt, include these msg files in the add_message_file() call as follows:

```
add_message_files( DIRECTORY msg
  FILES
  Message1.msg
  Message2.msg
)
```

add/ensure generate_messages(DEPENDENCIES …) is called

Using SRV

in ./srv
add/ensure that 'message_generation' is part of the find_package(catkin REQUIRED COMPONENTS ...) call in the CMakeLists.txt file (usually at the start of the file). (Note: *find_package(catkin REQUIRED COMPONENTS ...) cascades through the build. But it is risky not to include it in each package's CmakeLists.txt file*.)
In CMakeList.txt, include these msg files in the add_message_file() call as follows:

```
add_message_files( DIRECTORY msg
  FILES
  Service1.srv
  Service2.srv
)
```

MSG & SRV

add/ensure generate_messages(DEPENDENCIES std_msgs …) is called

FOLLOW UP
http://wiki.ros.org/catkin/CMakeLists.txt
http://wiki.ros.org/catkin/Tutorials/using_a_workspace#With_catkin_make

https://github.com/fairlight1337/ros_service_examples/