melodic | noetic | *Show EOL distros:* ☐

See rosbash on index.ros.org for more info including aything ROS 2 related. (https://index.ros.org/p/rosbash)

Documentation Status

ros (/ros?distro=noetic)*: mk (/mk?distro=noetic) | rosbash | rosboost_cfg (/rosboost_cfg?distro=noetic) | rosbuild (/rosbuild?distro=noetic) | rosclean (/rosclean?distro=noetic) | roscreate (/roscreate?distro=noetic) | roslang (/roslang?distro=noetic) | roslib (/roslib?distro=noetic) | rosmake (/rosmake?distro=noetic) | rosunit (/rosunit?distro=noetic)*

Package Links

- **Code API (http://docs.ros.org/en/noetic/api/rosbash/html)**
- Tutorials (/rosbash/Tutorials)
- FAQ (http://answers.ros.org/questions/scope:all/sort:activity-desc/tags:rosbash/page:1/)
- Changelog (http://docs.ros.org/en/noetic/changelogs/rosbash/changelog.html)
- Change List (/ros/ChangeList)
- Reviews (/rosbash/Reviews)

Dependencies (2)
Used by (8)
Jenkins jobs (10)

Package Summary

✓ Released **✓ Continuous Integration: 103 / 103** **✓ Documented**

Assorted shell commands for using ros with bash.

- Maintainer status: maintained
- Maintainer: Michel Hidalgo <michel AT ekumenlabs DOT com>, Jacob Perron <jacob AT openrobotics DOT org>
- Author: Jeremy Leibs, Thibault Kruse, Dirk Thomas <dthomas AT openrobotics DOT org>
- License: BSD
- Source: git https://github.com/ros/ros.git (https://github.com/ros/ros) (branch: noetic-devel)

Contents

1. Overview

The `rosbash` package contains some useful bash functions and adds tab-completion to a large number of the basic ros utilities.

When you source the distro specific setup file like:

```
source /opt/ros/noetic/setup.bash
```

you will implicitly get all bash-specific command.

2. Other Shells

The `rosbash` package includes limited support for `zsh` and `tcsh` by way of sourcing the `roszsh` or `rostcsh` files respectively. We currently do not provide documentation on these shells, though much of the functionality is similar to the bash shell extensions.

2.1 fish

Fish support has been added to rosbash. To activate it:

```
source /opt/ros/noetic/share/rosbash/rosfish
```

you can add this statement to the `conf.d` directory to execute it automatically on login, such as `~/.config/fish/conf.d/rosfish.fish`.

Sourcing a catkin workspace requires using bass (https://github.com/edc/bass):

```
bass source ~/catkin_ws/devel/setup.bash
```

You can have this done automatically as soon as you `cd` to the workspace directory, by adding a further config file (e.g. `~/.config/fish/conf.d/catkin.autosource.fish`):

```
function catkinSource --on-variable PWD
    status --is-command-substitution; and return
    if test -e ".catkin_workspace"; or test -e ".catkin_tools"
        bass source devel/setup.bash
        echo "Configured the folder as a workspace"
    end
end
```

3. Command line utilities

`rosbash` includes the following command line utilities:

- roscd (/rosbash#roscd) - change directory starting with package, stack, or location name
- rospd (/rosbash#rospd) - `pushd` equivalent of `roscd`
- rosd (/rosbash#rosd) - lists directories in the directory-stack
- rosls (/rosbash#rosls) - list files of a ros package
- rosed (/rosbash#rosed) - edit a file in a package
- roscp (/rosbash#roscp) - copy a file from a package
- rosrun (/rosbash#rosrun) - run executables of a ros package

3.1 roscd

`roscd` allows you to change directories using a package name, stack name, or special location.

Usage:

```
roscd <package-or-stack>[/subdir]
```

For example:

```
roscd roscpp
```

You can continue to use a relative path after the package name to go further into the package:

```
roscd roscpp/include/ros
```

`roscd` without argument will take you to `$ROS_WORKSPACE`.

Additionally, the `ROS_LOCATIONS` environment variable can be used to add additional special locations for use with `roscd`. `ROS_LOCATIONS` is a colon-separated list of `key=path` pairs.

For example, adding the following to your `.bashrc` file:

```
export ROS_LOCATIONS="pkgs=~/ros/pkgs:dev=~/ros/dev"
```

Will then allow you to type:

```
$ roscd dev
```

and end up in `~/ros/dev`.

3.2 rospd

`rospd` is the `pushd` equivalent of `roscd`. It allows you to keep multiple locations in a directory-stack while still using ros package names. You can then use the number of any directory in your directory stack to jump back there.

For example:

```
leibs@bar:~$ rospd hokuyo_node/
0 ~/ros/pkgs/laser_drivers/hokuyo_node
1 ~
leibs@bar:~/ros/pkgs/laser_drivers/hokuyo_node$ rospd roscpp_tutorials/
0 ~/ros/pkgs/ros_tutorials/roscpp_tutorials
1 ~/ros/pkgs/laser_drivers/hokuyo_node
2 ~
leibs@bar:~/ros/pkgs/ros_tutorials/roscpp_tutorials$ rospd laser_pipeline/
0 ~/ros/dev/laser_pipeline
1 ~/ros/pkgs/ros_tutorials/roscpp_tutorials
2 ~/ros/pkgs/laser_drivers/hokuyo_node
3 ~
leibs@bar:~/ros/dev/laser_pipeline$ rospd 1
0 ~/ros/pkgs/ros_tutorials/roscpp_tutorials
1 ~/ros/pkgs/laser_drivers/hokuyo_node
2 ~
3 ~/ros/dev/laser_pipeline
```

3.3 rosd

`rosd` lists the directories in your directory stack. This is for use with `rospd`.

For example:

```
leibs@bar:~/ros/pkgs/laser_drivers/hokuyo_node$ rosd
0 ~/ros/pkgs/laser_drivers/hokuyo_node
1 ~
2 ~/ros/dev/laser_pipeline
```

3.4 rosls

`rosls` allows you to view the contents of a package, stack, or location.

For example:

```
$ rosls roscpp
$ rosls roscpp/include/ros
```

3.5 rosed

`rosed` allows you to easily edit files in a ROS package by typing the package name and the name of the file you want to edit:

```
$ rosed roscpp_tutorials add_two_ints_server.cpp
```

Note: you can specify ANY file in a package, including those further down within the file hierarchy. If you specify an ambiguous file you will be prompted to select one.

For example:

```
$ rosed roscpp CMakeLists.txt
You have chosen a non-unique filename, please pick one of the following:
1) ~/ros/ros/core/roscpp/test/CMakeLists.txt
2) ~/ros/ros/core/roscpp/CMakeLists.txt
3) ~/ros/ros/core/roscpp/src/CMakeLists.txt
4) ~/ros/ros/core/roscpp/src/libros/CMakeLists.txt
#?
```

The default editor for rosed is vim. To use a different editor, set the `EDITOR` environment variable. E.g., in your ~/.bashrc:

```
export EDITOR='emacs -nw'
```

This example makes emacs the default editor.

You also can change the editor for one `rosed` call on the fly:

```
EDITOR=geany rosed rosbash rosbash
```

3.6 roscp

`roscp` allows you to conveniently copy a file from a package. Similar to `rosed` you can specify any file in the package regardless of hierarchy.

For example:

```
$ roscp roscpp_tutorials talker.cpp .
```

Will end up copying the file from `~/ros/pkgs/ros_tutorials/roscpp_tutorials/talker/talker.cpp`

3.7 rosrun

`rosrun` allows you to run an executable in an arbitrary package from anywhere without having to give its full path or `cd`/`roscd` there first.

Usage:

```
rosrun <package> <executable>
```

Example:

```
rosrun roscpp_tutorials talker
```

It's also possible to pass a `~parameter` using the following syntax (replace the ~ with an _):

```
rosrun package node _parameter:=value
```

Example:

```
rosrun my_package my_node _my_param:=value
```

...or run with a given name:

```
rosrun package node __name:=name
```

Example:

```
rosrun my_package my_node __name:=my_name
```

For more information about remapping, see: Remapping Arguments (https://wiki.ros.org/Remapping%20Arguments)

Starting in **Indigo**, rosrun has a `--prefix` option which can be used to run a node in gdb or valgrind.

Example:

```
rosrun --prefix 'gdb -ex run --args' my_package my_node
```

For more example prefixes, see: Roslaunch Nodes in Valgrind or GDB (http://wiki.ros.org/roslaunch/Tutorials/Roslaunch%20Nodes%20in%20Valgrind%20or%20GDB)

4. Tab Completion

`rosbash` enables tab-completion for its own tools and for a number of other ros utilities: rosmake (/rosmake), roslaunch (/roslaunch), rosparam (/rosparam), rosnode (/rosnode), rostopic (/rostopic), rosservice (/rosservice), rosmsg (/rosmsg), rossrv (/rossrv), rosbag (/rosbag).

There are a couple of generic completion rules that may be appropriate for other utilities. Looking through the `rosbash` source may provide insights into replicating similar functionality for other nodes, however, the code is not yet nicely generalized or re-usable. In a future release of ROS, we plan to incorporate a more generically re-usable tab-completion framework into rosbash and the other common shells used by ROS.