**Linux Ubuntuで**

**ROSプログラミング**

赤井 直紀

ver. 1 2020/11/03

1. **はじめに**

　本書の目的は，**Linux Ubuntuを始めて利用する人が最速でROSでプログラム（C/C++）を書けるようにすること**です．そのため，Linuxでプログラムを書くために最低限必要な知識とコマンド，そしてROSのプログラムが一体どの様になっているかの解説だけ行います．

1. **ROSのインストール**

　ROSのインストール方法は，Webに多数掲載されていますので，そちらを参考に行ってください（例えば<https://choreonoid.org/ja/manuals/latest/ros/install-ros.html>）．

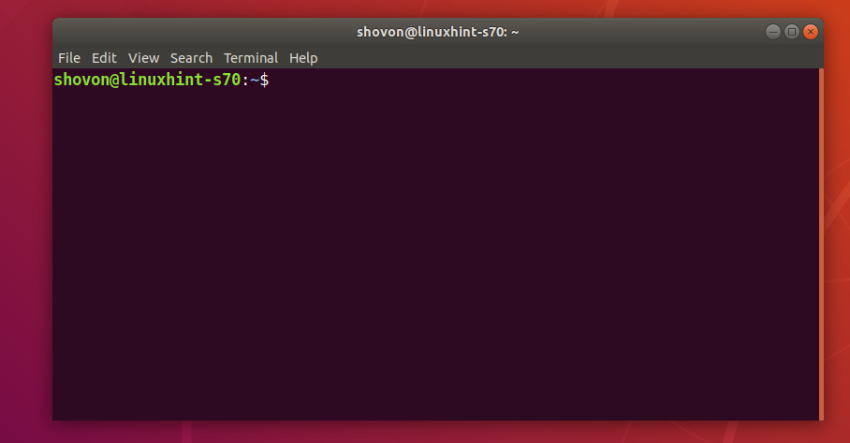
1. **Linuxのコマンド**

　まず，Linuxでプログラムを書くために最低限必要な知識とコマンドを説明します．

**３．１．端末**

　Linuxでプログラムを書く（というかLinuxを使う）ためには，**端末**というものに慣れる必要があります．端末とは，コマンドを打ってコンピュータを操作するためのインターフェースです．

　端末を起動するためには，Dashホームから「端末」もしくは「terminal」と打ちます．もしくは，「Ctrl + Alt + t」を押して開きます（端末は何度も開くのでショートカットコマンドで開く様にすべきです）．下の様な画面が端末です．



**３．２．ディレクトリの作成・削除とディレクトリ構造**

　Linux（WindowsもMacも）にはファイルを格納するための**ディレクトリ**があります．ディレクトリとは，ファイルなどを格納するフォルダのことです．このディレクトリが階層的になっているのですが，これを**ディレクトリ構造**と呼びます． Linuxを端末で扱うときはこの構造を理解しないといけません．以下，基本的なディレクトリ作成・削除などのコマンドを通して，ディレクトリ構造について説明します．

　まず端末を開くと「home」というディレクトリにいます．ここで

$ ls

というコマンドを打ってみます．すると，端末上に文字が出てきます．「ls」は「list」の省略で，「今いるディレクトリ（カレントディレクトリ）内のファイルやディレクトリを表示する」というコマンドです．lsはおそらくLinuxを使っていて最も使用するコマンドです．「何かコマンドを打ったらlsを打つ」という様に心掛けておくと良いです．

　端末を開いた最初はhomeディレクトリにいます．そこで，

$ mkdir catkin\_ws

と打ってください．「mkdir」は「make directory」の略であり，これで「catkin\_ws」というディレクトリが作成できます（catkin\_wsはROSで良く出てくる言葉です．細かい説明は置いておいて，まずは本書の通りの名前でディレクトリを作成していってください）．

　次に，作成したディレクトリに

$ cd catkin\_ws

で移動します．「cd」は「change directory」の意味で，その通りディレクトリ間を移動するために使われます（cdはlsについで使用するコマンドだと思います）．ここでさらに

$ mkdir src test

と入力し，「src」と「test」という名前のディレクトリを作成します．そうすると，以下の図に示す様な階層的なディレクトリ構造ができあがります．端末上ではcdを用いてディレクトリに間を移動するので，この様なディレクトリの構造が理解できていないと，目的のファイルにアクセスできなくなってしまいます．



　なお，

$ cd ..

と打つと，1つ上のディレクトリに戻ります（src内で打てばcatkin\_wsに戻ります）．また，

$ cd

とだけ入力すると，homeディレクトリに戻ります．

　今回は，catkin\_ws内に作成したtestというディレクトリは使用しませんので，最後にこれの削除の仕方を説明します．catkin\_ws内にcdで移動した後に

$ rm -r test

と打つとtestディレクトリを削除することができます．「rm」は「remove」の省略であり，ファイルやディレクトリを削除する際に利用します．「-r」はディレクトリを削除する際に必要なオプションです．Linuxのコマンドではこのように「-」を用いてコマンドにオプションを渡すことができます．

**３．３．ファイルの作成・編集**

　最後に，ファイル（ファイルとは.txtといった文章とかを書くものです）の作成・編集方法を説明します．まずcatkiin\_ws内に移動してください．なお，

$ pwd

と入力すると，今いるディレクトリの場所を端末に表示することができます．catkin\_wsに移動後，そこで，

$ gedit make.sh

と打ってください．「gedit」とは，ファイル編集ソフトであり，上記コマンドは「geditでmake.shという名前のファイルを開く」という意味です（ファイル編集ソフトは多数ありますので，そのうち自分の好きなものを探してみてください）．このファイルの中を

# /bin/bash

catkin\_make -DCMAKE\_BUILD\_TYPE=Release

と編集してください．これらの意味の説明は後述しますが，これで保存（保存ボタンを押すか「Ctrl + s」）すれば，ファイルを作成することができます．ファイルを閉じた後に再度開きたくなったら，また同じディレクトリから「gedit make.sh」を入力します．このファイルを削除したい場合は「rm make.sh」で消せますが，このファイルは後で使用するのでまだ消さないでください．

　かなり大雑把ですが，以上で

1. 端末の立ち上げ
2. ディレクトリ作成・削除，およびディレクトリ間の移動
3. ファイルの作成・編集，および削除

ができるようになりました．これができれば，プログラムの作成はできます．

1. **ROSパッケージの作成**

　ここからROSを使ってプログラミングするためにやることを説明します．

**４．１．workspaceの初期化**

　最初に作成した「catkin\_ws」とは，ROSでプログラミングする際に良く用いられるworkspaceの名前です．workspaceとは，プログラムの開発を行う場所の様なものです．まずはcatkin\_ws内のsrcに入り，以下のコマンドを実行してください（括弧内の言葉は端末上に打つ必要はありません）．

$ cd（cdだけ打って最初にhomeディレクトリに移動，念のため）

$ cd catkin\_ws/src（2つ以上のディレクトリ移動も1回のcdでできる）

$ catkin\_init\_workspace

src内で「catkin\_init\_workspace」を打つことで，catkin\_wsがROSのworkspaceとして機能する様になります．src内に「CMakeLists.txt」が生成されていれば成功です．確認してみましょう．

$ ls

これでCMakeLists.txtがあれば大丈夫です．

　次に，ROSの**パッケージ**を作成します．パッケージは，実際にROSのプログラムの単位となるようなものです．まず，catkin\_ws内のsrcに移動し，パッケージ作成のコマンドを打ちます．

$ catkin\_create\_pkg test\_pkg roscpp rospy std\_msgs

「catkin\_create\_pkg」がROSパッケージを作るコマンドで，その後ろがパッケージの名前（今回はtest\_pkg）です．それ以降に続くのは，どのROSのパッケージを新たに作成したパッケージ内で使うかです（今回はroscpp，rospy，std\_msgsを使用しますが，これらの詳細は省きます）．cdでtest\_pkgに入りlsで中身を見ると，すでにいくつかのファイルがあることが確認できます．

$ cd test\_pkg

$ ls

この中には，さらに「src」があることがわかります．実際のプログラムは，このsrc内で開発していきます（catkin\_ws内にsrcがあり，その中にパッケージがあり，さらにその中にsrcがあります．実際のプログラム開発は後者のsrcで行います．ややこしいですが間違えない様にしてください）．

**４．２．ROSでHello world**

「Hello world」とはプログラム界で良く用いられる超単純プログラムで，「Hello world」と表示するだけのプログラムを作ることです．実際にROSでこのプログラムを作成してみます．まず，test\_pkg内のsrcに入り，「hello\_world.cpp」というファイルを作成してください．

$ cd test\_pkg/src

$ gedit hello\_world.cpp

このファイルを以下の様に編集してください．

1. #include <ros/ros.h>
2. int main(int argc, char \*\*argv) {
3. ros::init(argc, argv, “hello\_world”);
4. ros::NodeHandle nh;
5. printf(“Hello world\n”);
6. return 0;
7. }

1行目でROSのプログラムを作成するために必要なヘッダファイルをインクルードします．3行目からこのプログラムのメイン関数が始まります．4，5行目はROSのプログラムで必要なおまじないです（「おまじない」とはプログラム用語で良く用いられる，「理由はわからなくても良いからとりあえず書いておく」という様なものです）．6行目でHello worldを出力します．7行目で0を返して終了します（6行目の¥はLinuxだとバックスラッシュです）．編集ができたらファイルを保存して閉じます（保存は「Ctrl + s」で！）．

　次に，これをコンパイル（実行できる様に変換すること）します．test\_pkg内に戻り，CMakeLists.txtを編集します（catkin\_wsの下のsrc何にもCMakeLists.txtが存在しますが，これではありません！注意しましょう）．

$ cd ..

$ gedit CMakeLists.txt

CMakeLists.txtの一番下に，以下を追加してください．

add\_executable(hello\_world src/hello\_world.cpp)

target\_link\_libraries(hello\_world ${catkin\_LIBRARIES})

「add\_executable(hello\_world src/hello\_world.cpp)」は，srcの内にあるhello\_world.cppをhello\_worldという名前で実行ファイルにコンパイルするという意味です．また「target\_link\_libraries(hello\_world ${catkin\_LIBRARIES})」は，hello\_worldという実行ファイルにcatkin\_LIBRARIESをリンクするという意味です（catkin\_LIBRARIESはROSのライブライの様なものです）．これが記述できたらCMakeLists.txtを閉じ，再びcatkin\_wsに戻り，プログラムをコンパイルします．

$ cd（homeディレクトリに戻る方が早い）

$ cd catkin\_ws

$ sh make.sh

「sh make.sh」は，make.shという**シェルスクリプト**を**シェル**として実行するというコマンドです．要するに，「make.shの中身のコマンドを実行する」という意味です．実際にmake.shの中身に何があるかを確認すると

catkin\_make -DCMAKE\_BUILD\_TYPE=Release

が記述してあります．このコマンドは，catkin\_ws内にあるROSパッケージ全てをコンパイルしてくれます（適切にパッケージ内のCMakeLists.txtを修正する必要はあります）．なお，make.sh内の1行目に書いてある

# /bin/bash

は，make.shをシェルスクリプトとして認識させるためのおまじないです．

　コンパイルが完了すると，「devel」というディレクトリができ，その下に「setup.bash」が作成されます．setup.bashはこのディレクトリに格納されているプログラムの情報を保有している様なものであり，今開いている端末にこの情報を理解させる必要があります（端末はそれぞれ設定が異なることに注意してください）．そのために，catkin\_ws内で

$ source devel/setup.bash

と入力して，setup.bashの情報を端末に読み込ませます．これにより，端末がcatkin\_ws内に存在するコンパイルされたプログラムの存在を知ることができます．

　実際にROSのプログラムを実行するにあたり，もう1つ別の端末を立ち上げてください（端末の立ち上げは「Ctrl + Alt + t」）．そこで，

$ roscore

と打ってください．これはROSのプロセスを管理するメインプログラムを立ち上げるものです．ROSで開発したプログラムを起動させるには，必ずroscoreが動いている必要があります．その後，devel/setup.bashを読み込んだ端末に戻り，

$ rosrun test\_pkg hello\_world

と入力することで，作成したHello worldのプログラムを実行できます．「rosrun」は，ROSのプログラムを実行するという意味です．続く「test\_pkg hello\_world」は，「test\_pkg内に存在するhello\_worldという実行ファイルを実行する」という意味です．これで端末に「Hello world」と表示されるはずです．

**４．３．ROS内での通信（PublisherとSubscriber）**

　ROSの特徴は，「プログラム（ノード）間でメッセージのやり取りを行うこと」です．例えばセンサのデータや，何かしら処理を行った結果がメッセージとして扱われます．1つのノードが処理した結果をメッセージとして送信すると，他の全てのノードもそのメッセージを受信することができます．そのため，1つのノードに複雑な機能を実装するのではなく，役割単位でノードを作成していくことが多いです．なお，ROSで行われるこのメッセージの送信，受信のことをPublish，Subscribeと呼んでおり，またそれを行うノードのことをPublisher，Subscriberと呼びます．

**４．４．Publisherの実装**

　具体的なPublisherを実装してみます．まず，test\_pkg内のsrcに入ります（実際に開発するプログラムはパッケージ内のsrcに配置します）．そこで，test\_pub.cppというファイルを作成します．

$ cd

$ cd catkin\_ws/src/test\_pkg/src

$ gedit test\_pub.cpp

test\_pub.cppを以下の様に編集します．

1. #include <ros/ros.h>
2. #include <std\_msgs/Header.h>
3. int main(int argc, char \*\*argv) {
4. ros::init(argc, argv, "test\_pub");
5. ros::NodeHandle nh("~");
6. ros::Publisher headerPub = nh.advertise<std\_msgs::Header>("/test\_msg", 1);
7. ros::Rate loopRate(10.0);
8. while (ros::ok()) {
9. ros::spinOnce();
10. std\_msgs::Header header;
11. header.stamp = ros::Time::now();
12. headerPub.publish(header);
13. printf("Publish time is %lf\n", header.stamp.toSec());
14. loopRate.sleep();
15. }
16. return 0;
17. }

詳細はサンプルプログラム内で解説しているので省きます．このファイルを保存し（保存は「Ctrl + s」），再度test\_pkg内のCMakeLists.txtを編集します．

$ cd ..

$ gedit CMakeLists.txt

再度CMakeLists.txtの一番下に以下を追加します．

add\_executable(test\_pub src/test\_pub.cpp)

target\_link\_libraries(test\_pub ${catkin\_LIBRARIES})

これで，catkin\_ws内のmake.shを実行すれば，test\_pub.cppをコンパイルすることができます．このプログラムの実行は後程行うので，次はSubscriberを実装しましょう．

**４．５．Subscriberの実装**

　Publisherの実装と同様に，test\_pkgのsrc内で，test\_sub.cppを作成します．

$ cd src

$ gedit test\_sub.cpp

このファイルを以下の様に編集します．

1. #include <ros/ros.h>
2. #include <std\_msgs/Header.h>
3. void headerCB(const std\_msgs::Header::ConstPtr &msg) {
4. printf("Subscribe time is %lf\n", msg->stamp.toSec());
5. }
6. int main(int argc, char \*\*argv) {
7. ros::init(argc, argv, "test\_sub");
8. ros::NodeHandle nh("~");
9. ros::Subscriber headerSub = nh.subscribe("/test\_msg", 1, headerCB);
10. ros::spin();
11. return 0;
12. }

これも詳細はサンプルプログラム内に書かれているので，詳細は割愛します．このプログラムをコンパイルするために，再度test\_pkg内のCMakeLists.txtを編集します．

$ cd ..

$ gedit CMakeLists.txt

CMakeLists.txtに以下を追加します．

add\_executable(test\_sub src/test\_sub.cpp)

target\_link\_libraries(test\_sub ${catkin\_LIBRARIES})

これでtest\_sub.cppがコンパイルできるようになります．catkin\_wsに戻り，test\_pub.cppとtest\_sub.cppをコンパイルしてみます．

$ cd

$ cd catkin\_ws

$ sh make.sh

エラー無くコンパイルできれば完了です．

**４．６．PublisherとSubscriberの実行**

　コンパイルしたプログラムを実際に実行してみます．端末にコンパイルされたプログラムを認識させるためには，catkin\_ws内のdevel内にあるsetup.bashを読み込む必要がありました（1ど読み込んだ端末で何度も読み込む必要はありませんが，念のためです）．

$ source devel/setup.bash

そしてPublisherを実行します（違う端末でroscoreを立ち上げておかなければならないことを忘れないでください）．

$ rosrun test\_pkg test\_pub

端末に「Publish time is …」という様に時間が表示されます．ここでもう1つ端末を立ち上げて（Ctrl + Alt + t），再度catkin\_wsに入り，Subscriberを実行します．

$ cd catkin\_ws

$ source devel/setup.bash（新しい端末なのでsetup.bashを読み込まないといけません）

$ rosrun test\_pkg test\_sub

Subscriberを実行した端末に「Subscribe time is …」と表示されます．これは，Publisherが送信したメッセージをSubscriberで受信し表示したものです．以上でROSを使った最低限のプログラミングができました．

1. **まとめ**

　本書では，詳細はさておき，Linux UbuntuでROSを用いたプログラミングを行う方法を説明しました．かなり大雑把ですが，Ubuntuで

1. 端末を使う
2. ディレクトリの操作，ディレクトリ間の移動
3. ファイルの操作

を習得できたはずです．最低限これだけ理解できれば，プログラムを書くことはできます．また，実際にROSを用いてPublisherとSubscriberを実装しました．これにより，

1. メッセージを送信するプログラムの実装
2. メッセージを受信するプログラムの実装

ができる様になりました．ROSでの全てのプログラムは，この2つが基本になります．より高度なプログラムを書くためには，使用するメッセージの種類や，使用するライブラリを増やしていく必要があります．