# [ffmpeg第五弹：Qt+SDL+ffmpeg视频播放演示](https://mp.weixin.qq.com/s/Z0w8KGfYzxLOOWc8yCXpHw)

1. 前言

在前几篇文章当中，有提到过用源码去搭建ffmpeg的命令环境开发，为啥要这样去搭建环境，为什么不用直接用下面这个命令在ubuntu下安装多快，简单又方便：

sudo apt install ffmpeg

今天分享ffmepg第五弹：ffmpeg+qt+SDL的真正开发环境，就要用源码安装的方式去在qt里面调用ffmpeg相关的库；还记得之前源码搭建创建的三个文件夹不：

bin   ffmpeg\_sources    ffmpeg\_build

**bin文件夹下是编译得到的二进制文件**

txp@txp-virtual-machine:~/bin$ ls ffmpeg  ffplay  ffprobe  lame  nasm    
ndisasm  x264

**ffmpeg\_sources文件下是下载的各种库的源码：**

txp@txp-virtual-machine:~/ffmpeg\_sources$ ls  
fdk-aac                  lame-3.100.tar.gz      SDL2-2.0.14.tar.gz  
ffmpeg                 libvpx          SVT-AV1  
ffmpeg-4.2.1              nasm-2.14.02         x264  
ffmpeg-4.2.1.tar.bz2      nasm-2.14.02.tar.bz2   x265\_git  
ffmpeg-snapshot.tar.bz2   opus  
lame-3.100                SDL2-2.0.14

**ffmpeg\_build文件夹主要是ffmpeg的一些库文件，等下下面演示的模板就要调用ffmpeg相关的库：**

txp@txp-virtual-machine:~/ffmpeg\_build/lib$ ls  
cmake         libmp3lame.a          libSDL2.la        libswscale.a  
libavcodec.a    libmp3lame.la          libSDL2main.a     libvpx.a  
libavdevice.a   libopus.a             libSDL2main.la    libx264.a  
libavfilter.a   libopus.la            libSDL2.so        libx265.a  
libavformat.a   libpostproc.a           libSDL2\_test.a    pkgconfig  
libavutil.a     libSDL2-2.0.so.0        libSDL2\_test.la  
libfdk-aac.a    libSDL2-2.0.so.0.14.0   libSvtAv1Enc.a  
libfdk-aac.la   libSDL2.a             libswresample.a

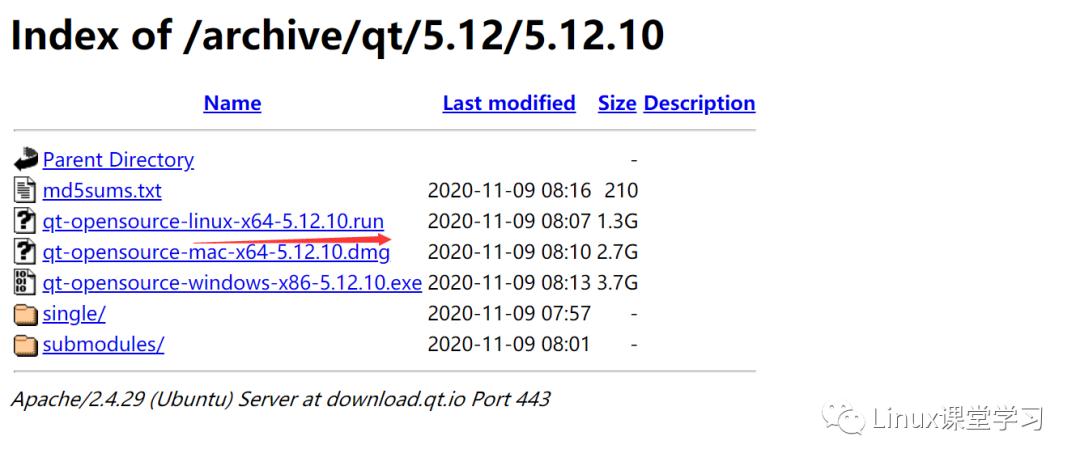
因为我已经搭建好了开发环境，从现在来看的话，如果直接用命令去安装ffmpeg的话，到时候我们在qt的环境中去调用ffmpeg的库，至少到目前为止我暂时不知道去如何配置相关路径来调用ffmpeg的库；所以我们明白了这点，那么就撸起袖子肝就是。

1. qt环境搭建

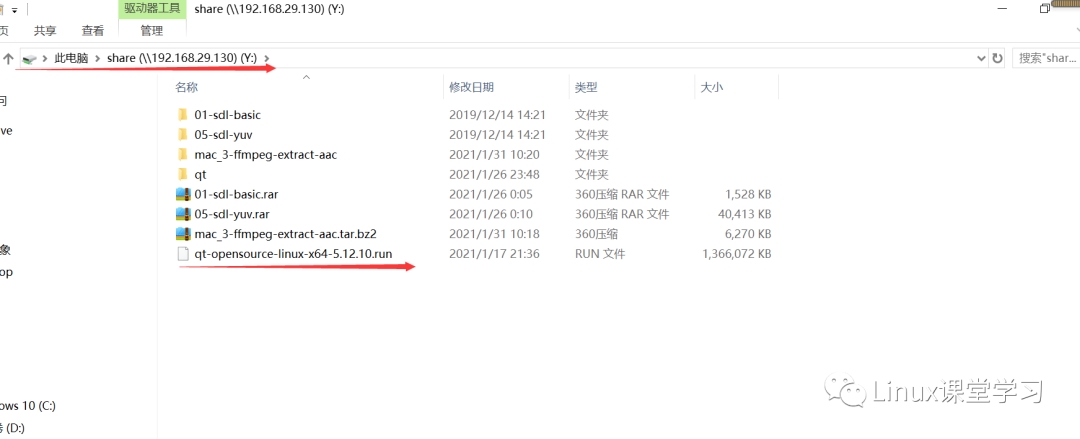
玩过qt的朋友，对于这块应该比我熟悉多了；不过有可能有一些朋友可能没有接触过qt的话，为此我还是简单演示一下qt的安装步骤：

首先我们要去qt的官网下载linux版本的源码安装包：

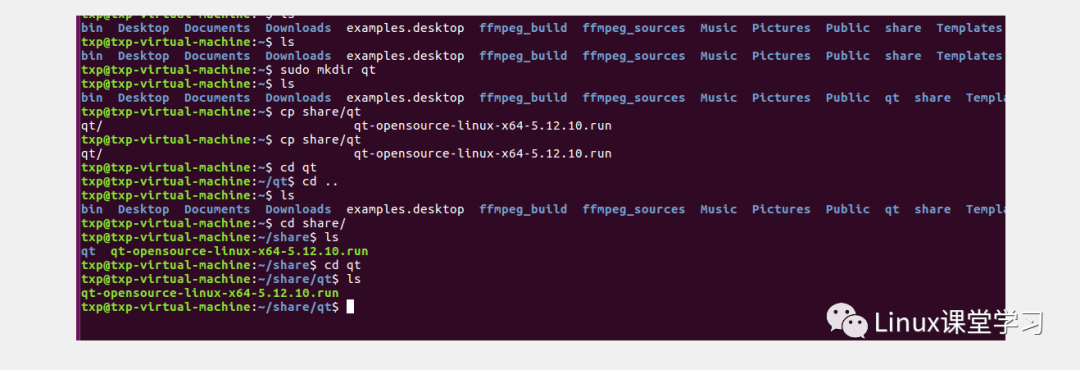
版本下载地址：  
<https://download.qt.io/archive/qt/5.12/5.12.10/>

开始安装qt：

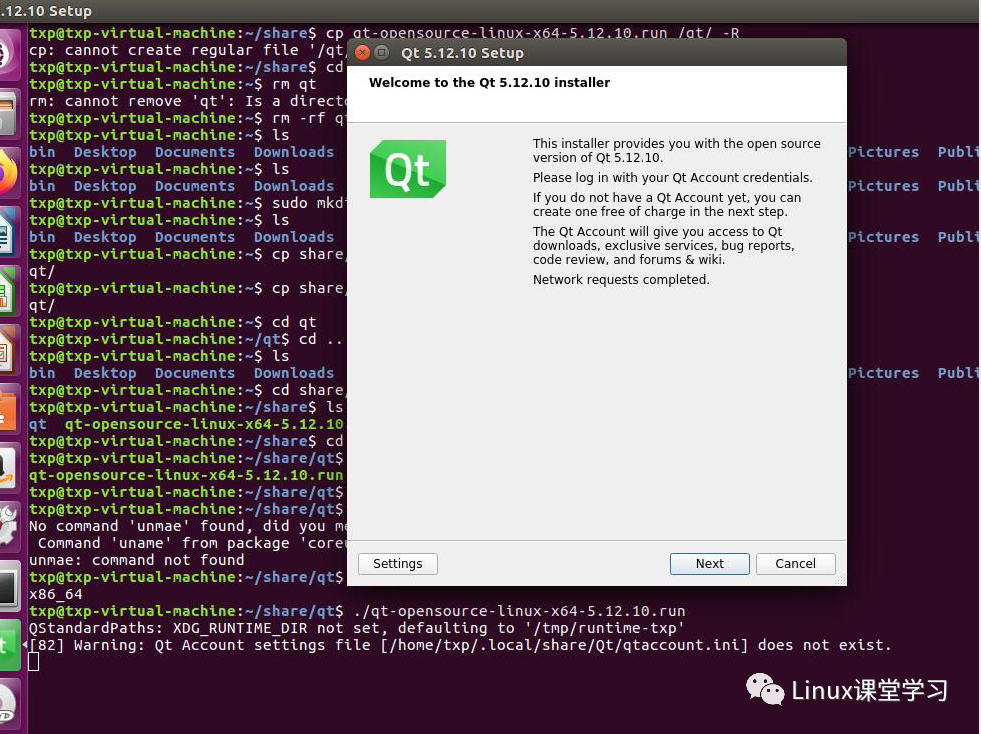
这里我直接把qt的源码包下载到samba服务共享文件下，当然你也可以直接在ubuntu下载：

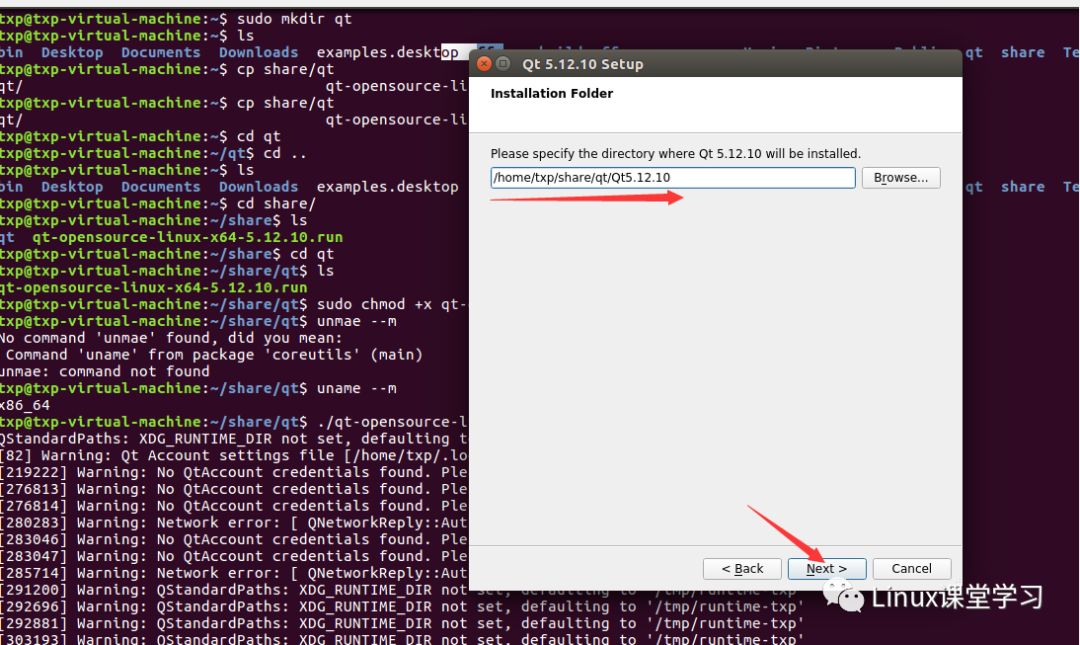


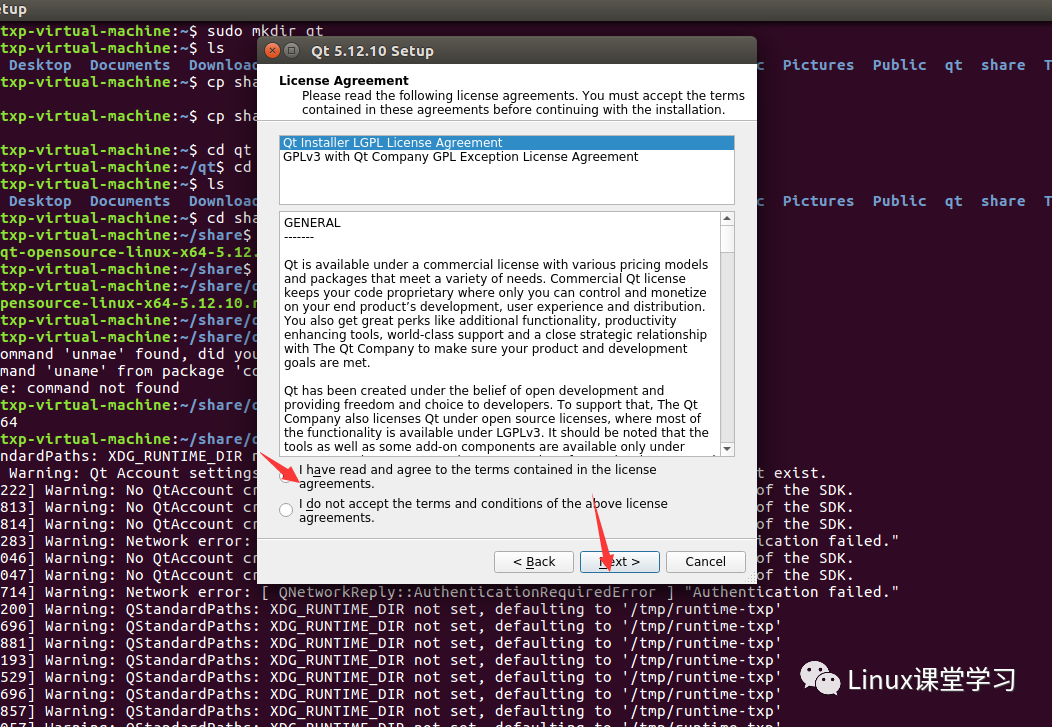
然后我在share目录创建一个qt文件夹，用存放qt安装的地方，然后直接运行这个源码文件：



接着就会出现qt的安装界面：







最终qt就安装完成了，但是如果你运行qt执行失败的话(注意qtcreator的所放在的路线)：

txp@txp-virtual-machine:~/share/qt/Qt5.12.10/Tools/QtCreator/bin$ sudo ./qtcreator

Got keys from plugin meta data ("xcb")  
QFactoryLoader::QFactoryLoader() checking directory path "/home/txp/share/qt/Qt5.12.10/Tools/QtCreator/bin/platforms" ...

loaded library "/home/txp/share/qt/Qt5.12.10/Tools/QtCreator/lib/Qt/plugins/platforms/libqxcb.so"

loaded library "Xcursor"  
Segmentation fault (core dumped)

**解决qt运行失败的方法：**

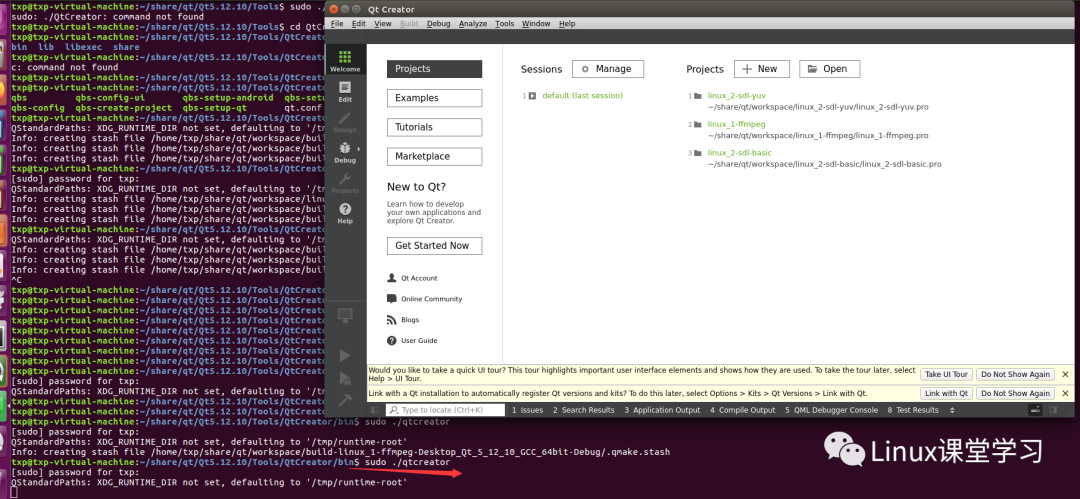
在这个脚本里面添加一句话：



然后接着再安装相关插件：

sudo apt install --reinstall libxcb-xinerama0

然后我们执行一下，就可以成功打开qt了：



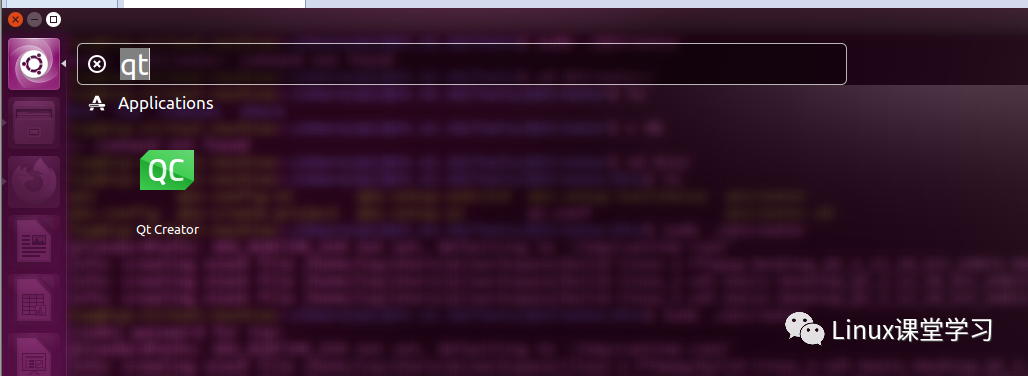
还有一种方法直接打开qt，因为按照上面的这种方式打开的话，每次都要跑到这个目录去执行这条语句才行：

txp@txp-virtual-machine:~/share/qt/Qt5.12.10/Tools/QtCreator/bin$   
sudo ./qtcreator

现在我们只要执行下面这条语句就不用这么麻烦了：

sudo chown -R txp:txp ~/.config/

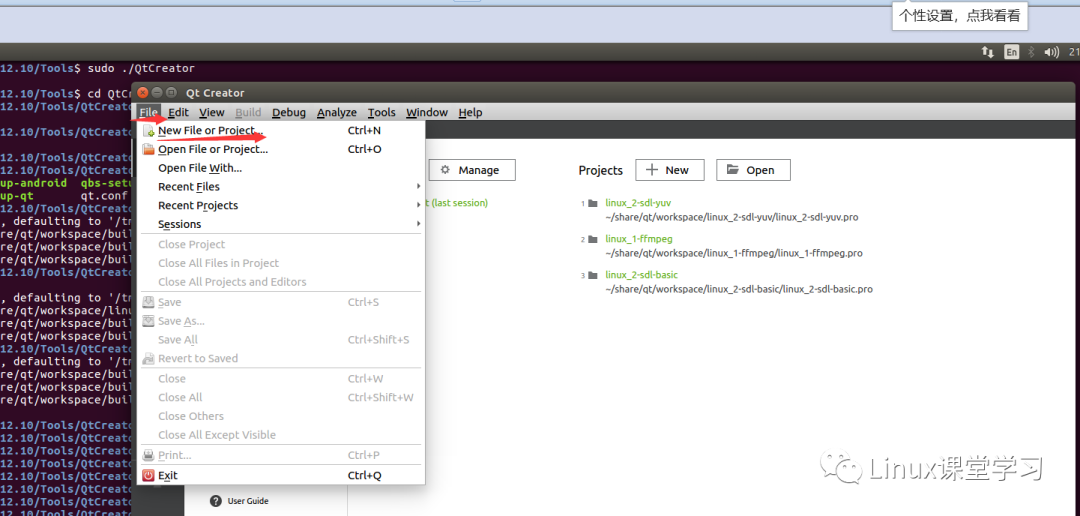
然后你就可以像在windows环境下去直接打开这个软件就行：

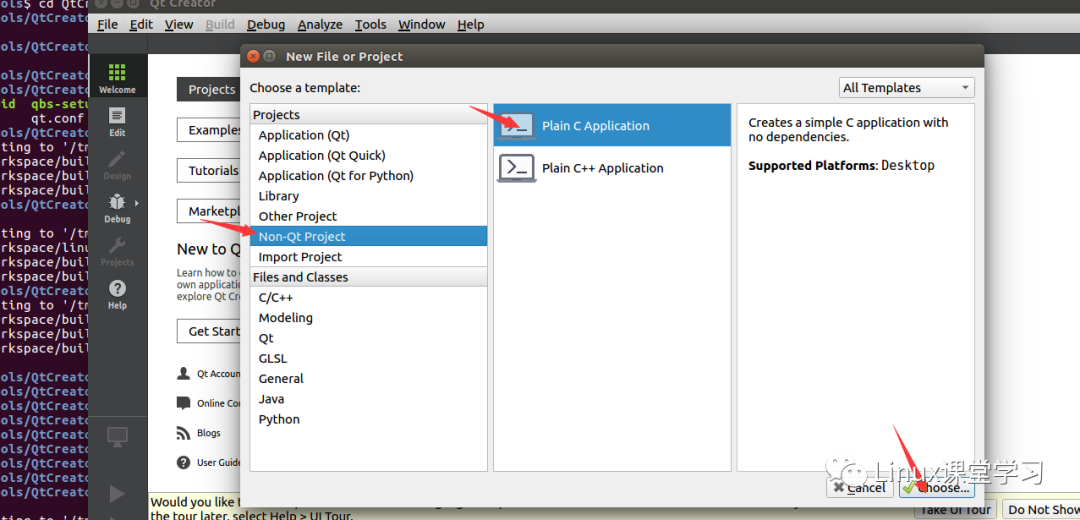


1. 牛刀小试，调用ffmpeg库

### **3.1 示例模板，显示打印ffmpeg的版本：**

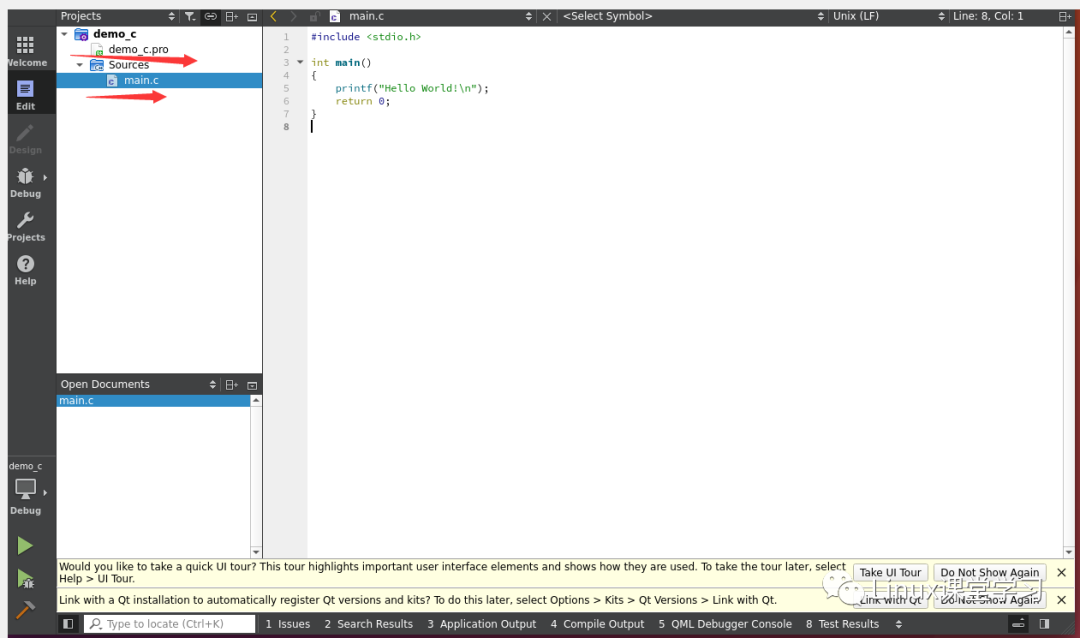
现在我们打开刚才安装好的qt软件，来创建一个工程：







最终一个工程项目就建立好了：



我们可以看到两个文件，一个是以.pro结尾的qt工程管理配置文件，一个主源码文件，现在我们就简单使用ffmpeg库来打印ffmpeg的版本号：

配置pro文件：pro文件里面默认是：

TEMPLATE = app  
CONFIG += console  
CONFIG -= app\_bundle  
CONFIG -= qt  
  
SOURCES += \  
        main.c

现在我们要加入ffmpeg\_build目录下的ffmpeg库文件：

TEMPLATE = app  
CONFIG += console  
CONFIG -= app\_bundle  
CONFIG -= qt  
  
SOURCES += \  
        main.c  
INCLUDEPATH += /home/txp/ffmpeg\_build/include  
#LIBS += /home/txp/ffmpeg\_build/lib/libSDL2.so  
  
LIBS += /home/txp/ffmpeg\_build/lib/libavcodec.a \  
        /home/txp/ffmpeg\_build/lib/libavdevice.a \  
        /home/txp/ffmpeg\_build/lib/libavfilter.a \  
        /home/txp/ffmpeg\_build/lib/libavformat.a \  
        /home/txp/ffmpeg\_build/lib/libavutil.a \  
        /home/txp/ffmpeg\_build/lib/libswresample.a \  
        /home/txp/ffmpeg\_build/lib/libswscale.a

main.c源码文件就可以修改成这样：

#include <stdio.h>  
//包含ffmpeg头⽂件  
#include "libavutil/avutil.h"  
int **main**()  
{  
    printf("Hello FFMPEG, version is %s\n", av\_version\_info());  
    return 0;  
}

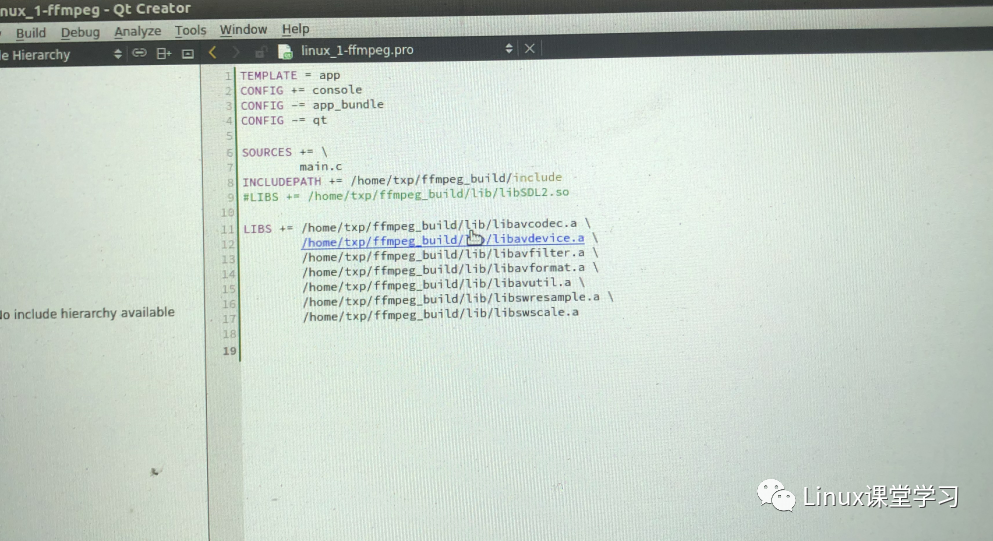
最终结果，显示ffmpeg的版本为 4.2.1：

20:01:19: Starting /home/txp/share/qt/workspace/build-linux\_1-ffmpeg-Desktop\_Qt\_5\_12\_10\_GCC\_64bit-Debug/linux\_1-ffmpeg ...

Hello FFMPEG, version is 4.2.1

20:01:19: /home/txp/share/qt/workspace/build-linux\_1-ffmpeg-Desktop\_Qt\_5\_12\_10\_GCC\_64bit-Debug/linux\_1-ffmpeg exited with code 0

注意：我们的ubuntu运行环境一定要是64位的，不然安装不了这个版本的qt;还有一点，其实我们在.pro文件里面，按键盘上的ctrl键然后把鼠标放到库文件路径上，是可以打开里面的：



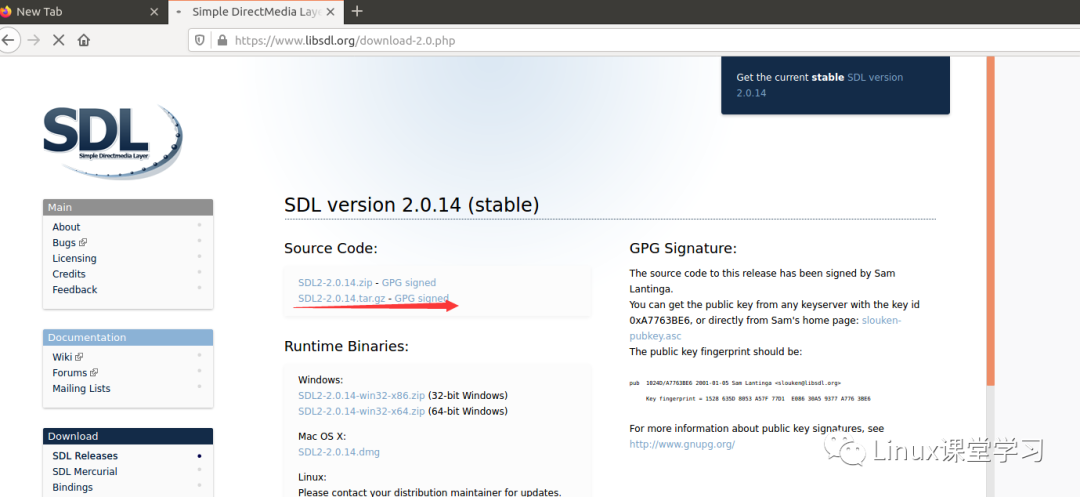
3.2、搭建SDL，然后播放yuv格式的视频文件：

关于什么是SDL，这里我就不造轮子了，可以参考雷神的文章介绍：

<https://blog.csdn.net/leixiaohua1020/article/details/11954039>



下面我们去SDL的官网下源码包就行安装：



开始安装，先把这个源码包放到ffmpeg\_sources目录下去，然后进行解压：

txp@txp-virtual-machine:~/ffmpeg\_sources$ tar zxf SDL2-2.0.14.tar.gz   
txp@txp-virtual-machine:~/ffmpeg\_sources$ ls  
fdk-aac                  lame-3.100.tar.gz     SDL2-2.0.14.tar.gz  
ffmpeg                  libvpx                SVT-AV1  
ffmpeg-4.2.1              nasm-2.14.02          x264  
ffmpeg-4.2.1.tar.bz2      nasm-2.14.02.tar.bz2  x265\_git  
ffmpeg-snapshot.tar.bz2   opus  
lame-3.100                SDL2-2.0.14

然后执行：

txp@txp-virtual-machine:~/ffmpeg\_sources/SDL2-2.0.14$ ./autogen.sh   
Generating build information using autoconf  
This may take a **while** ...  
Now you are ready to run ./configure

这里提示了你直接运行 ./configure:

./configure --prefix=/home/txp/ffmpeg\_build --bindir=/home/txp/bin

然后进行编译：make -j4

txp@txp-virtual-machine:~/ffmpeg\_sources/SDL2-2.0.14$ make -j4

最后再执行sudo make install就行，SDL就安装成功了：

txp@txp-virtual-machine:~/ffmpeg\_sources/SDL2-2.0.14$ sudo make install

下面我再创建一个工程，具体过程我就再写了，和第一个工程创建是一样的：这里我的main.c文件里面的源代码，大家先不用管代码具体啥意思：

#include <stdio.h>  
#include <string.h>  
#include "SDL2/SDL.h"//包含SDL动态库文件  
  
//自定义消息类型  
#define REFRESH\_EVENT   (SDL\_USEREVENT + 1)     // 请求画面刷新事件  
#define QUIT\_EVENT      (SDL\_USEREVENT + 2)     // 退出事件

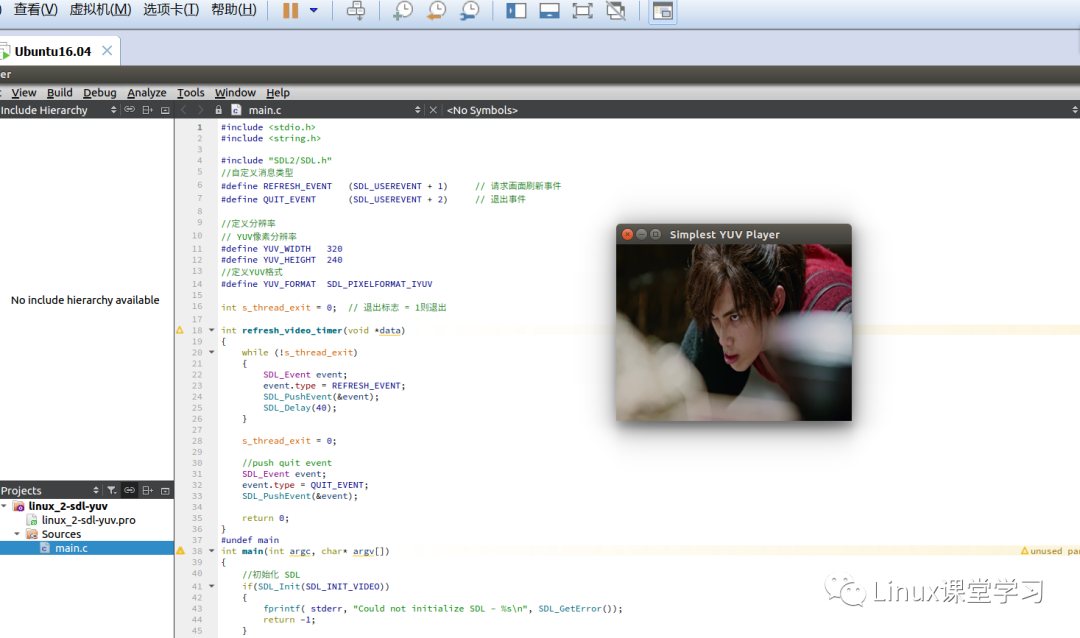
//定义分辨率  
// YUV像素分辨率  
#define YUV\_WIDTH   320  
#define YUV\_HEIGHT  240  
//定义YUV格式  
#define YUV\_FORMAT  SDL\_PIXELFORMAT\_IYUV

int s\_thread\_exit = 0;  // 退出标志 = 1则退出  
int refresh\_video\_timer(void \*data)  
{  
    **while** (!s\_thread\_exit)  
    {  
        SDL\_Event event;  
        event.type = REFRESH\_EVENT;  
        SDL\_PushEvent(&event);  
        SDL\_Delay(40);  
    }  
    s\_thread\_exit = 0;  
    //push quit event  
    SDL\_Event event;  
    event.type = QUIT\_EVENT;  
    SDL\_PushEvent(&event);  
    return 0;  
}  
#undef main  
int main(int argc, char\* argv[])  
{  
    //初始化 SDL  
    **if**(SDL\_Init(SDL\_INIT\_VIDEO))  
    {  
        fprintf( stderr, "Could not initialize SDL - %s\n", SDL\_GetError());  
        return -1;  
    }  
    // SDL  
    SDL\_Event event;                            // 事件  
    SDL\_Rect rect;                              // 矩形  
    SDL\_Window \*window = NULL;                  // 窗口  
    SDL\_Renderer \*renderer = NULL;              // 渲染  
    SDL\_Texture \*texture = NULL;                // 纹理  
    SDL\_Thread \*timer\_thread = NULL;            // 请求刷新线程  
    uint32\_t pixformat = YUV\_FORMAT;            // YUV420P，即是SDL\_PIXELFORMAT\_IYUV  
    // 分辨率  
    // 1. YUV的分辨率  
    int video\_width = YUV\_WIDTH;  
    int video\_height = YUV\_HEIGHT;  
    // 2.显示窗口的分辨率  
    int win\_width = YUV\_WIDTH;  
    int win\_height = YUV\_WIDTH;  
    // YUV文件句柄  
    FILE \*video\_fd = NULL;  
    const char \*yuv\_path = "yuv420p\_320x240.yuv";  
    size\_t video\_buff\_len = 0;  
    uint8\_t \*video\_buf = NULL; //读取数据后先把放到buffer里面  
    // 我们测试的文件是YUV420P格式  
    uint32\_t y\_frame\_len = video\_width \* video\_height;  
    uint32\_t u\_frame\_len = video\_width \* video\_height / 4;  
    uint32\_t v\_frame\_len = video\_width \* video\_height / 4;  
    uint32\_t yuv\_frame\_len = y\_frame\_len + u\_frame\_len + v\_frame\_len;  
    //创建窗口  
    window = SDL\_CreateWindow("Simplest YUV Player",  
                           SDL\_WINDOWPOS\_UNDEFINED,  
                           SDL\_WINDOWPOS\_UNDEFINED,  
                           video\_width, video\_height,  
                           SDL\_WINDOW\_OPENGL|SDL\_WINDOW\_RESIZABLE);  
    **if**(!window)  
    {  
        fprintf(stderr, "SDL: could not create window, err:%s\n",SDL\_GetError());  
        goto \_FAIL;  
    }  
    // 基于窗口创建渲染器  
    renderer = SDL\_CreateRenderer(window, -1, 0);  
    // 基于渲染器创建纹理  
    texture = SDL\_CreateTexture(renderer,  
                                pixformat,  
                                SDL\_TEXTUREACCESS\_STREAMING,  
                                video\_width,  
                                video\_height);  
  
    // 分配空间  
    video\_buf = (uint8\_t\*)malloc(yuv\_frame\_len);  
    **if**(!video\_buf)  
    {  
        fprintf(stderr, "Failed to alloce yuv frame space!\n");  
        goto \_FAIL;  
    }  
  
    // 打开YUV文件  
    video\_fd = fopen(yuv\_path, "rb");  
    **if**( !video\_fd )  
    {  
        fprintf(stderr, "Failed to open yuv file\n");  
        goto \_FAIL;  
    }  
    // 创建请求刷新线程  
    timer\_thread = SDL\_CreateThread(refresh\_video\_timer,  
                                    NULL,  
                                    NULL);  
  
    **while** (1)  
    {  
        // 收取SDL系统里面的事件  
        SDL\_WaitEvent(&event);  
  
        **if**(event.type == REFRESH\_EVENT) // 画面刷新事件  
        {  
            video\_buff\_len = fread(video\_buf, 1, yuv\_frame\_len, video\_fd);  
            **if**(video\_buff\_len <= 0)  
            {  
                fprintf(stderr, "Failed to read data from yuv file!\n");  
                goto \_FAIL;  
            }  
            // 设置纹理的数据 video\_width = 320， plane  
            SDL\_UpdateTexture(texture, NULL, video\_buf, video\_width);  
  
            // 显示区域，可以通过修改w和h进行缩放  
            rect.x = 0;  
            rect.y = 0;  
            float w\_ratio = win\_width \* 1.0 /video\_width;  
            float h\_ratio = win\_height \* 1.0 /video\_height;  
            // 320x240 怎么保持原视频的宽高比例  
            rect.w = video\_width \* w\_ratio;  
            rect.h = video\_height \* h\_ratio;  
//            rect.w = video\_width \* 0.5;  
//            rect.h = video\_height \* 0.5;  
  
            // 清除当前显示  
            SDL\_RenderClear(renderer);  
            // 将纹理的数据拷贝给渲染器  
            SDL\_RenderCopy(renderer, texture, NULL, &rect);  
            // 显示  
            SDL\_RenderPresent(renderer);  
        }  
        **else** **if**(event.type == SDL\_WINDOWEVENT)  
        {  
            //If Resize  
            SDL\_GetWindowSize(window, &win\_width, &win\_height);  
            printf("SDL\_WINDOWEVENT win\_width:%d, win\_height:%d\n",win\_width,  
                   win\_height );  
        }  
        **else** **if**(event.type == SDL\_QUIT) //退出事件  
        {  
            s\_thread\_exit = 1;  
        }  
        **else** **if**(event.type == QUIT\_EVENT)  
        {  
            break;  
        }  
    }  
  
\_FAIL:  
    s\_thread\_exit = 1;      // 保证线程能够退出  
    // 释放资源  
    **if**(timer\_thread)  
        SDL\_WaitThread(timer\_thread, NULL); // 等待线程退出  
    **if**(video\_buf)  
        free(video\_buf);  
    **if**(video\_fd)  
        fclose(video\_fd);  
    **if**(texture)  
        SDL\_DestroyTexture(texture);  
    **if**(renderer)  
        SDL\_DestroyRenderer(renderer);  
    **if**(window)  
        SDL\_DestroyWindow(window);  
  
    SDL\_Quit();  
  
    return 0;  
  
}

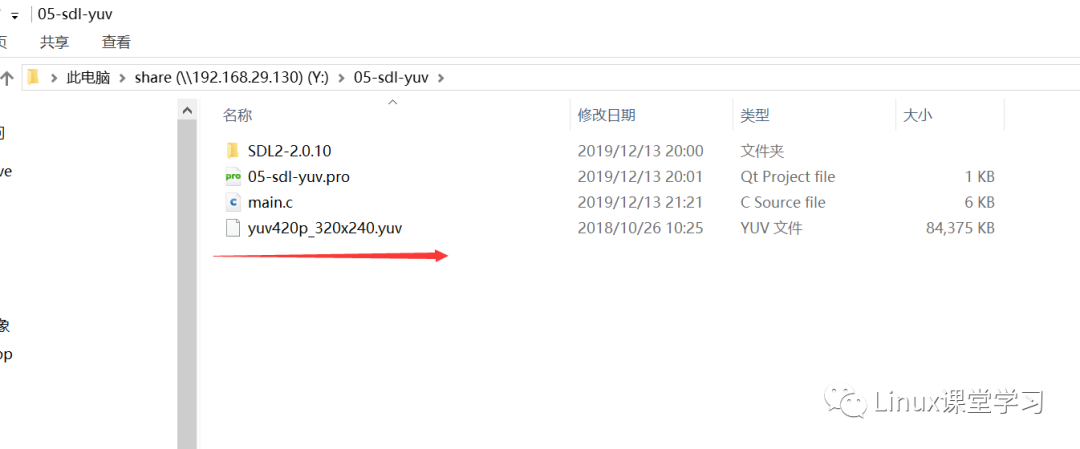
.pro文件配置成：

TEMPLATE = app  
CONFIG += console  
CONFIG -= app\_bundle  
CONFIG -= qt  
  
SOURCES += \  
        main.c  
INCLUDEPATH += /home/txp/ffmpeg\_build/include  
LIBS += /home/txp/ffmpeg\_build/lib/libSDL2.so

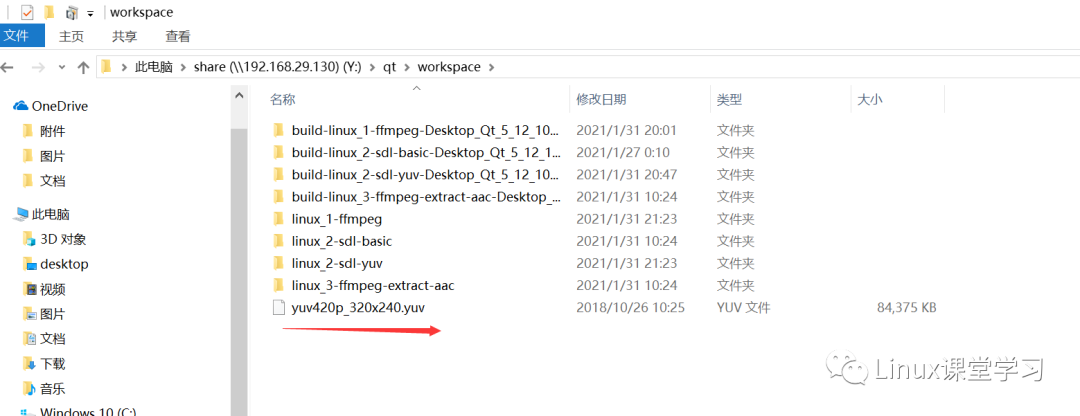
最终运行结果就可以看到在播放一个yuv格式的视频文件了：



注：这里的播放yuv格式的视频文件，我是事先已经准备好的：



同时要注意我们要把播放的视频文件放到工程目录下，不然播放是不会成功的：



1. 总结

现在ffmpeg真正的开发环境已经搭建完了，其实上面搭建环境蛮折腾人的，特别是源码安装ffmpeg。好了今天的文章就分享到这里了，如果你在看完文章后，有不懂的地方可以后台私聊我。