**目录：**

[1.firefox，关闭最后的标签时，同时关闭浏览器 2](#__RefHeading___Toc564_155741408)

[2.Firefox无法同步Linux版和Windows版的配置及收藏等 2](#__RefHeading___Toc566_155741408)

[3.Office word的图片由“上下型”改成“嵌入型”嵌入文字中 2](#__RefHeading___Toc568_155741408)

[4.对数组名进行取地址操作 2](#__RefHeading___Toc570_155741408)

[5.Resnet50中的7.71GOPs是如何得到的 2](#__RefHeading___Toc572_155741408)

[6.一维数组二维化 2](#__RefHeading___Toc574_155741408)

[7.FC的scale是干什么用的 2](#__RefHeading___Toc576_155741408)

[8.pair<float, int>的用法 2](#__RefHeading___Toc578_155741408)

[9.Makefile中使用shell命令 4](#__RefHeading___Toc582_155741408)

[10.OpenCV的Mat数据类型 6](#__RefHeading___Toc584_155741408)

[11.C中的char bgr递减到0 6](#__RefHeading___Toc586_155741408)

[12.OpenCV可以用Mat\_<int>/Mat\_<double>等类型 6](#__RefHeading___Toc874_955763420)

[13.#if的使用 6](#__RefHeading___Toc878_955763420)

[14.C++中形参使用引用 7](#__RefHeading___Toc882_955763420)

[15.openCV的IplImage 7](#__RefHeading___Toc888_955763420)

[16.常用的五个函数(I/O) 8](#__RefHeading___Toc894_955763420)

[2. Constexpr 9](#__RefHeading___Toc898_955763420)

[3.C的文件操作 10](#__RefHeading___Toc902_955763420)

[4.linux中文件名包含有“-”符号 10](#__RefHeading___Toc906_955763420)

[6.VIM中的特殊字符 11](#__RefHeading___Toc1045_1477830120)

[7. Linux下getopt()函数的简单使用 13](#__RefHeading___Toc2202_1399078383)

[8. linux的find命令于shell命令组合使用 13](#__RefHeading___Toc1560_125025805)

[9. c++中的path.c\_str()函数 15](#__RefHeading___Toc1562_125025805)

1. **firefox，关闭最后的标签时，同时关闭浏览器**

* 在地址栏输入:about:config。打开火狐配置中心
* 搜索栏中输入：browser.tabs.closeWindowWithLastTab
* 将默认的的值由原来的true改为false，即可完成。
* 将true改为false的方法是双击条目即可。

1. **Firefox无法同步Linux版和Windows版的配置及收藏等**

由于中国防火墙的原因。

1. **Office word的图片由“上下型”改成“嵌入型”嵌入文字中**

文件—选项—高级—剪切、复制粘贴，下面的“将图像插入/粘贴为”

1. **对数组名进行取地址操作**

Int a={0,1,2,3,4,5,6,7,8,9};

Int\* ptr=\*(&a+1);

Printf(“a[0]=:%d, \*ptr=%d \n”, a[0], \*ptr);

1. **Resnet50中的7.71GOPs是如何得到的**
2. **一维数组二维化**

Int \*pp\_arr=new int[Nrow \* MAZCol];

Int (\*p)[MAXCol]=( int (\*)[MAXCol] )pp\_arr;

这样pp\_arr就可以当成二维数组使用了，pp\_arr[i][j]。

1. **FC的scale是干什么用的**

float convScale = dpuGetOutputTensorScale(taskFC, FC\_OUT\_NODE, 0);

是一个比例系数，从addr和api分别得到的数据相差一个**scale。**

1. **pair<float, int>的用法**

pair定义在utility中，它组合两个类型数据到一个数据中，两个数据类型可以一样，也可以不一样。

**pair提供的操作：**

* + pair<T1,T2> p1;
  + pair<T1,T2> p1(v1,v2);
  + make\_pair(v1,v2);
  + p1 < p2;
  + p1 == p2;
  + p.first;
  + p.second;

**pair的引用：**

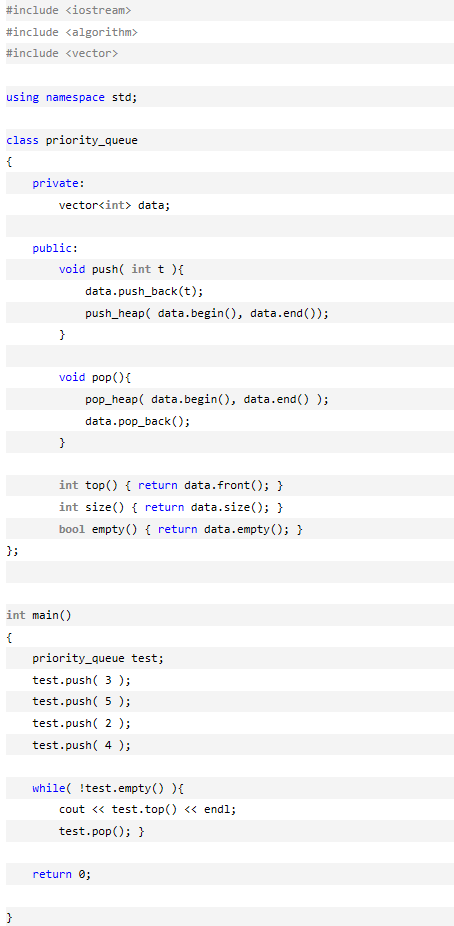
pair名加“.”加first/second进行使用。

xxx.first xxx.second.

**9.**

priority\_queue相对于queue的不同之处在于：优先队列实现了内部自动排序，可根据自己需要自定义排序规则，可以自己编写函数或者仿函数用于内部优先级的确定。

priority\_queue调用 STL里面的 make\_heap(), pop\_heap(), push\_heap() 算法实现，也算是堆的另外一种形式。先写一个用 STL 里面堆算法实现的与真正的STL里面的 priority\_queue用法相似的priority\_queue， 以加深对 priority\_queue 的理解:



STL里面的 priority\_queue 写法与此相似，只是增加了模板及相关的迭代器什么的。 priority\_queue 对于基本类型的使用方法相对简单。他的模板声明带有三个参数:

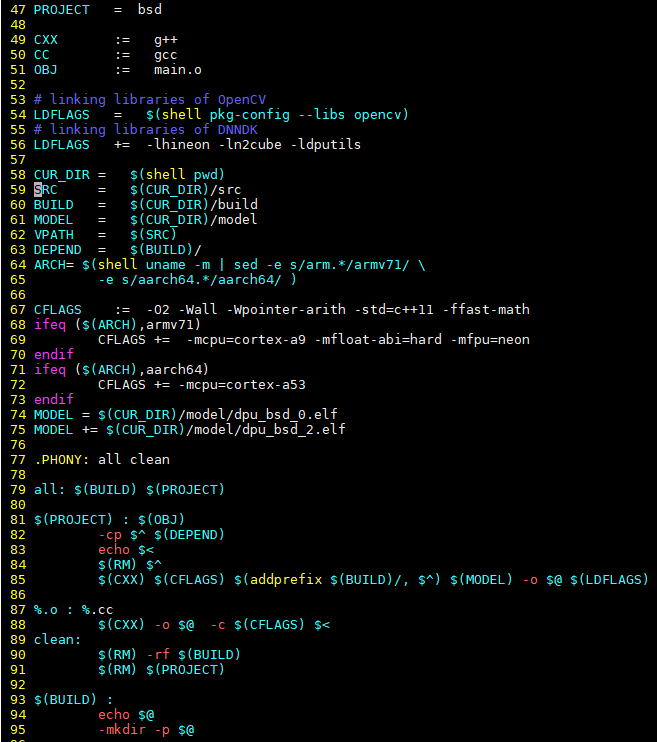
priority\_queue<Type, Container, Functional>  
其中Type 为数据类型， Container 为保存数据的容器，Functional 为元素比较方式。

Container 必须是用数组实现的容器，比如 vector, deque 但不能用 list.

STL里面默认用的是 vector. 比较方式默认用 operator< , 所以如果你把后面俩个参数缺省的话，优先队列就是大顶堆，队头元素最大。

1. **Makefile中使用shell命令**

Makefile中是用shell命令方法：**CUR\_PATH = $(shell pwd)**



ARCH= $(shell uname -m | sed -e s/arm.\*/armv71/ **\**

-e s/aarch64.\*/aarch64/ )

**CFLAGS** := -O2 -Wall -Wpointer-arith -std=c++11 -ffast-math

ifeq ($(ARCH),armv71)

CFLAGS += -mcpu=cortex-a9 -mfloat-abi=hard -mfpu=neon

endif

ifeq ($(ARCH),aarch64)

CFLAGS += -mcpu=cortex-a53

endif

MODEL = $(CUR\_DIR)/model/dpu\_bsd\_0.elf

MODEL += $(CUR\_DIR)/model/dpu\_bsd\_2.elf

**.PHONY**: **all clean**

**all: $(BUILD) $(PROJECT)**

**$(PROJECT) : $(OBJ)**

-cp $^ $(DEPEND)

$(RM) $^

$(CXX) $(**CFLAGS**) $(**addprefix** $(BUILD)/, $^) $(MODEL) -o **$@** $(LDFLAGS)

**%.o : %.cc**

$(CXX) -o $@ -c $(CFLAGS) $<

**clean:**

$(RM) -rf $(BUILD)

$(RM) $(PROJECT)

**$(BUILD) :**

-mkdir -p $@

Echo $@: 这个是/root/Desktop/xilinx\_dnndk\_v2.08/ZCU104/samples/bsd/build

Echo $^: 这个是main.o (可能目前就一个依赖文件main.o)

Echo $<: 这个是main.o（目前就一个依赖文件）

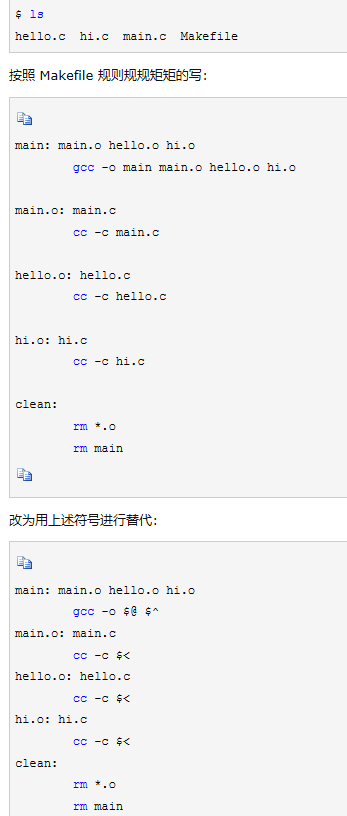
***跟下面的说明貌似不符合啊 ？？？***

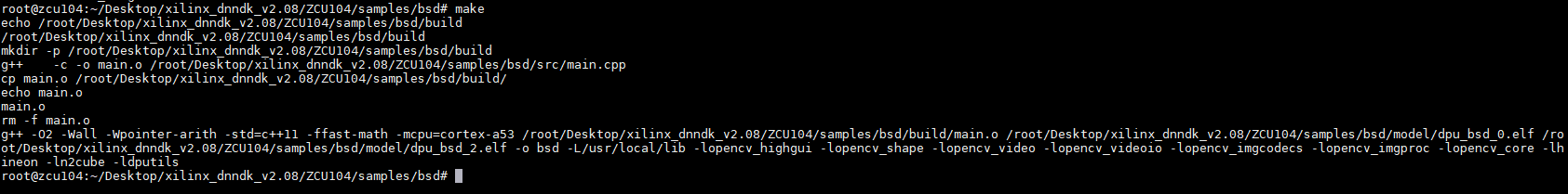
**$@  表示目标文件**

**$^  表示所有的依赖文件**

**$<  表示第一个依赖文件**

**$?  表示比目标还要新的依赖文件列表**





**问题**：

问题现象，是使用resnet50的Makefile当BSD的makefile，出现错误！main.o生成在了bsd/下面，即当前目录下。没有build/main.o这个obj文件，所以在编译$(PROJECT)时，所依赖的build/main.o没有，导致出错！

由此可以知道执行顺序：

**$(BUILD)**  **$(PROJECT)** 

1. **OpenCV的Mat数据类型**

Mat（图像数据结构体）代替了CvMat和IPIImage，并将功能模块化。

[Mat](https://link.jianshu.com/?t=http://docs.opencv.org/3.2.0/d3/d63/classcv_1_1Mat.html" \l "details)它由两个数据部分组成：**矩阵头**（包含矩阵尺寸，存储方法，存储地址等信息）和一个指向存储所有像素值的**矩阵**（根据所选存储方法的不同矩阵可以是不同的维数）的指针。矩阵的指针可以是xxx.ptr(i)，也可以是xxx.Data。

**xxx.ptr(i)是第i行的首元素指针。**

**Xxx.Data是图像数据的指针。**

**Xxx.at<Vec8>(row,col)[0/1/2]，代表了row行col列的B/G/R通道的数据。**

1. **C中的char bgr递减到0**

此处如果没有显示的声明bgr为signed char，则可能编译器会自动优化了bgr为无符号型数据，则无法判断是小于0的情况，形成死循环。

而且，注意在Linux中，变量的作用范围，如果在for循环中定义的局部变量，i。它的作用域只在{}大括号之间。即：

for(char i=0; i<3; i++)

{

//在此处有效。

}

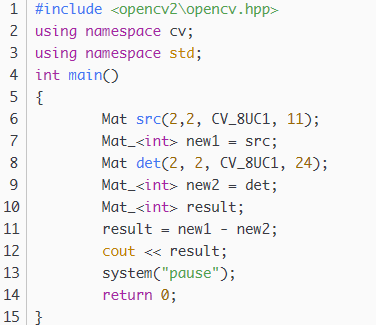
**i++; //此处的i已经不再作用域内了**。

1. **OpenCV可以用Mat\_<int>/Mat\_<double>等类型**

因为Amandeep的网络需要对输入的图片进行bias=-128的操作，输入图像的数据在BSD预处理的时候已经变成了负数。需要将Mat数据转换成可以进行负数运算的类型。

Mat\_<int>可以进行带负数的矩阵操作。

Mat类型针对的是图像数据的运算。



Mat类型是opencv2.0后的类型，使用此类型无需进行内存管理，即无需手动分配内存，在不需要时自动释放内存，但因目前的许多嵌入式系统仅支持c语言，故而除了在某些嵌入式系统中使用cvMat或IplImage, 基本使用Mat类型。

1. **#if的使用**

#if后面接的是常量表达式

#if ()此处括号内的内容(1 == x)这样可能无法得到正确的结果，需要的是一个常量表达式。

1. C++中形参使用引用

在函数的形参中使用引用，可以防止大量数据复制的开销。(其实引用就是指针，所以形参中只复制了1个指针地址).

但是使用引用的缺点是，可能修改原数据。所以在形参列表中可以添加一个**const**前缀进行修饰。防止修改原始数据值。

1. **openCV的IplImage**

IplImage结构

由于OpenCV主要针对的是计算机视觉方面的处理，因此在函数库中，最重要的结构体是IplImage结构。IplImage结构来源于Intel的另外一个函数库Intel Image Processing Library (IPL)，该函数库主要是针对图像处理。IplImage结构具体定义如下：

typedef struct \_IplImage

    {

        int nSize;         /\* IplImage大小 \*/

        int ID;            /\* 版本 (=0)\*/

        int nChannels;     /\* 大多数OPENCV函数支持1,2,3 或 4 个通道 \*/

        int alphaChannel; /\* 被OpenCV忽略 \*/

        int depth;         /\* 像素的位深度，主要有以下支持格式： IPL\_DEPTH\_8U, IPL\_DEPTH\_8S, IPL\_DEPTH\_16U,IPL\_DEPTH\_16S, IPL\_DEPTH\_32S,

IPL\_DEPTH\_32F 和IPL\_DEPTH\_64F \*/

        char colorModel[4]; /\* 被OpenCV忽略 \*/

        char channelSeq[4]; /\* 同上 \*/

        int dataOrder;     /\* 0 - 交叉存取颜色通道, 1 - 分开的颜色通道.

                               只有cvCreateImage可以创建交叉存取图像 \*/

        int origin;        /\*图像原点位置： 0表示顶-左结构,1表示底-左结构 \*/

        int align;         /\* 图像行排列方式 (4 or 8)，在 OpenCV 被忽略，使用 widthStep 代替 \*/

        int width;        /\* 图像宽像素数 \*/

        int height;        /\* 图像高像素数\*/

        struct \_IplROI \*roi; /\* 图像感兴趣区域，当该值非空时，

只对该区域进行处理 \*/

        struct \_IplImage \*maskROI; /\* 在 OpenCV中必须为NULL \*/

        void \*imageId;     /\* 同上\*/

        struct \_IplTileInfo \*tileInfo; /\*同上\*/

        int imageSize;     /\* 图像数据大小(在交叉存取格式下ImageSize=image->height\*image->widthStep），单位字节\*/

        char \*imageData;    /\* 指向排列的图像数据 \*/

        int widthStep;     /\* 排列的图像行大小，以字节为单位 \*/

        int BorderMode[4]; /\* 边际结束模式, 在 OpenCV 被忽略\*/

        int BorderConst[4]; /\* 同上 \*/

        char \*imageDataOrigin; /\* 指针指向一个不同的图像数据结构（不是必须排列的），是为了纠正图像内存分配准备的 \*/

    } IplImage;

IplImage结构体是整个OpenCV函数库的基础，在定义该结构变量时需要用到函数cvCreatImage，变量定义方法如下：

IplImage\* src="/cvCreateImage"(cvSize(400,300), IPL\_DEPTH\_8U,3);

上句定义了一个IplImage指针变量src，图像的大小是400×300，图像颜色深度8位，3通道图像。

1. 常用的五个函数(I/O)
   * 图像载入函数

函数cvLoadImage载入指定图像文件，并返回指向该文件的IplImage指针。函数支持bmp、jpg、 png、 tiff等格式的图像。其函数原型如下：

IplImage\* cvLoadImage( const char\* filename, int iscolor);

其中，filename 是待载入图像的名称，包括图像的扩展名；iscolor是一个辅助参数项，可选正数、零和负数三种值，正数表示作为三通道图像载入，零表示该图像作为单通道图像，负数表示载入图像的通道数由图像文件自身决定。

* + 窗口定义函数

函数cvNamedWindow定义一个窗口，用于显示图像。其函数原型如下：

int cvNamedWindow( const char\* name, unsigned long flags );

其中，name是窗口名，flags是窗口属性指标值，可以选择CV\_WINDOW\_AUTOSIZE和0两种值。CV\_WINDOW\_AUTOSIZE表示窗口尺寸与图像原始尺寸相同，0表示以固定的窗口尺寸显示图像。

* + 图像显示函数

函数cvShowImage是在指定的窗口中显示图像，其函数原型如下：

void cvShowImage( const char\* name, const CvArr\* image );

其中，name是窗口名称，image是图像类型指针，一般是IplImage指针。

* + 图像保存函数

函数cvSaveImage以指定的文件名保存IplImage类型的指针变量，其函数原型如下：

int cvSaveImage( const char\* filename, const CvArr\* image );

其中，filename是图像保存路径和名称，image是IplImage指针变量。

* + 图像销毁函数

函数cvReleaseImage销毁已定义的IplImage指针变量，释放占用内存空间。其函数原型如下：

void cvReleaseImage( IplImage\*\* image );

其中，image为已定义的IplImage指针。

1. Constexpr



1. C的文件操作

比较BSD两个文件的输出结果。

1. linux中文件名包含有“-”符号

Linux 中"-"有特殊的含义，比如在命令的后面加"-"表示后面跟的是选项。但是如何新建带名字中有"-"，下面介绍三种方法，比如要新建 文件 "-a"：

1、cd .>-a

2、vi -- -a

3、echo"">-a

上面三种方法都可以在当前目录下新建一个"-a" 文件 ，当然还可以有别的方法新建，这里就不在介绍。

如果要删除"-a"就比较麻烦了，如果你输入 rm -a，系统会提示你：

rm：无效选项 -- a

请尝试执行‘rm --help’来获取更多信息。

对于这样的 文件 可以有两种方法来删除，如下所示：

1、rm ./-a

2、rm -- -a

改名字也不用说了，基本上就是在 文件 名前面加"--"来表示转义字符就可以了！

1. **openCV头文件防止嵌套包含**

#ifndef OPENCV\_module\_name\_header\_name\_HPP

#define OPENCV\_module\_name\_header\_name\_HPP

namespace cv { namespace mynamespace {

...

}}

#endif

1. **VIM中的特殊字符**

Vim中可见特殊字符会直接显示，不可见特殊字符会显示为该字符在命令行的输入方式， 例如 \r 显示为 ^M。通过 :help digraph-table 可看到所有Vim中可输入的特殊字符：

char digraph hex dec official name

^@ NU 0x00 0 NULL (NUL)

^A SH 0x01 1 START OF HEADING (SOH)

^B SX 0x02 2 START OF TEXT (STX)

^C EX 0x03 3 END OF TEXT (ETX)

^D ET 0x04 4 END OF TRANSMISSION (EOT)

^E EQ 0x05 5 ENQUIRY (ENQ)

^F AK 0x06 6 ACKNOWLEDGE (ACK)

^G BL 0x07 7 BELL (BEL)

^H BS 0x08 8 BACKSPACE (BS)

^I HT 0x09 9 CHARACTER TABULATION (HT)

^@ LF 0x0a 10 LINE FEED (LF)

^K VT 0x0b 11 LINE TABULATION (VT)

^L FF 0x0c 12 FORM FEED (FF)

^M CR 0x0d 13 CARRIAGE RETURN (CR)

第一列为特殊字符，第二列为digraph（见下文），第三列为十六进制表示， 第四列为十进制表示，第五列为该字符的官方名称。所以如何查看文件中的这些字符呢？

" 显示隐藏字符

:set list

" 不显示隐藏字符

:set nolist

" 设置显示哪些隐藏字符

:set listchars=eol:$,tab:>-,trail:~,extends:>,precedes:<

也可以通过外部命令查看二进制表示。**转为二进制：:%!xxd，恢复：:%!xdd -r。**

**输入非可见字符：digraph**

Vim 中显然没有什么特殊字符选择工具，但提供了两种输入特殊字符的方式：

通过两个字符来输入一个特殊字符（digraph）。

直接通过编码值（ASCII或Unicode）输入。

其中digraph是一种类似双拼的方法，连续输入两个字符来表示一个特殊字符。 需要先按下前导键<Ctrl-K>，例如在编辑模式下输入：

<Ctrl-K>Rg

将会出现®字符，其中"Rg"是该字符的digraph（双拼）。 所有的digraph可以通过:help digraph-table查询。

**输入非可见字符：字符编码**

除了 digraph 外还可直接通过字符编码来输入它，这样我们即使没有输入法也可以输入中文。 这种方式也是在插入模式下进行的，需要先按下前导键<Ctrl-V>（Windows下<Ctrl-Q>）。 有下列5种方式：

十进制值ASCII：^Vnnn (000 <= nnn <= 255)

八进制值：^VOnnn 或 ^Vonnn (000 <= nnn <= 377)

十六进制值：^VXnn or ^Vxnn (00 <= nn <= FF)

十六进制BMP Unicode：^Vunnnn (0000 <= nnnn <= FFFF)

十六进制任何Unicode：^VUnnnnnnnn (00000000 <= nnnnnnnn <= 7FFFFFFF)

是时候打开 Vim 了，进入插入模式后依次按下以下字符：

<Ctrl-v>u6768<Ctrl-v>u73fa<Esc>

将会得到 杨珺 两个字符，没错这是 Harttle 的名。

**换行的搜索/替换**

如果你还不了解Vim的搜索和替换，请移步在 Vim 中优雅地查找和替换。

换行在Vim中的行为很特殊也不够一致，需要单独讨论一下。首先区分一下\r和\n：

* 前者是回车（Carriage Return），在Vim中可通过<c-k>CR输入，显示为^M。
* 后者是换行（New Line），在Vim中通过<CR>（回车）键输入，显示为回车并换行；

所以对于Windows风格换行（\r\n）在Vim中会在每行结尾显示^M。

**搜索**

在搜索模式（/）搜索换行时仍然应当使用\n字符，因为Vim的换行（Unix风格）确实是\n而不是\r\n。例如：

/foo\nbar

可以匹配到所有的：

foo

bar

**替换**

注意 :s 命令替换为换行（NL, \n）时，应当写 \r 而不是\n。 例如将所有逗号替换为换行：

:%s/,/\r/g

如果使用\n则目标会被替换为空字符NULL（显示为^@）。

将DOS风格换行（\r\n）的文件转为Unix风格换行（\n）其实很简单，不需要手动查找替换：

:set fileformat=unix

:w

Excel 等软件经常用 CR（^M, \r）来换行，怎么把 \r换成 \n 呢？

:%s/<Ctrl-k>cr/\r/g

7. [Linux下getopt()函数的简单使用](https://www.cnblogs.com/qingergege/p/5914218.html)

https://www.cnblogs.com/qingergege/p/5914218.html

首先需要包含的头文件，<**unistd.h>**

**涉及到的变量：**

extern char\* optarg；保存选项参数的。

extern int optind； 记录下一次检索位置。

extern int opterr； 是否将错误信息输出到stderr上。

extern int optopt； 不再选项字符串optstring中的选项。

**变量：optstring**

optstring就是在命令行中输入得选项字符串，optind检索得也是argv[n].

**optind是根据每次检测而变化的：**

optind，使用“s：”中得冒号，指示了s是有参数的。那么下一次检索的argv[n]就应该是argv[3]了。即optind=3；

8. linux的find命令于shell命令组合使用

\* {}是find 命令返回的结果；

\* \;是整个命令结束尾部的标记；

\* **注意问题：find命令需要夹 -type f，这样不会把当前目录的文件，作为ls的输入列出。**

**如：**find **.** -type f -mtime +5 **-exec** ls -l --time-style='+%Y-%m-%d' {} \;

中的“.”会作为ls的输入。





**9. c++中的path.c\_str()函数**

从命名上看到c\_str()，是为了兼容c没有string类型，而实现的一个函数。在c++中string是一STL中的一个类，包含了一定的成员函数。

**C++ 中的C\_str()函数用法**

语法:

const **char \***c\_str();

c\_str()函数返回一个指向正规C字符串的指针常量, 内容与本string串相同.

这是为了与c语言兼容，在c语言中没有string类型，故必须通过string类对象的成员函数c\_str()把string 对象转换成c中的字符串样式。

看看下面的例子，各位就会明白了

int main()

{

string path;

path="ansf";

cout<<path<<endl;;

printf("%s",path.c\_str());

return 0;

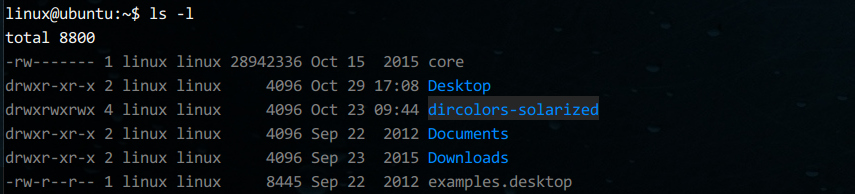
}

两次都是输出 anaf

10.stat()，lstat()，fstat() 获取文件/目录的相关信息

stat 的使用

Linux有个命令，ls -l,效果如下：



这个命令能显示文件的类型、操作权限、硬链接数量、属主、所属组、大小、修改时间、文件名。它是怎么获得这些信息的呢，请看下面的讲解。

stat 的基本使用

stat：返回一个与此命

需要包含的头文件： <sys/types.h>，<sys/stat.h>，<unistd.h>

函数原型：

　　int stat(const char \*path, struct stat \*buf);

int fstat(int fd, struct stat \*buf);

int lstat(const char \*path, struct stat \*buf);

参数：

　　对于stat() & lstat()来说path，是要查看属性的文件或目录的全路径名称

　　对于fstat，fd 是要查看属性文件的文件描述符

　　buf：指向用于存放文件属性的结构体，函数成功调用后，buf各个字段存放各个属性。

返回值 成：

　　功返回0；　　　　　　错误返回 -1；

给定一个文件：

　　stat 函数获得一个与此命名文件有关的信息（到一个struct stat 类型的buf中） 。

　　fstat 函数获得文件描述符 fd 打开文件的相关信息（到一个struct stat 类型的buf中） 。

　　lstat 函数类似于 stat，但是当命名文件是一个符号连接时， lstat 获取该符号连接的有关信息，而不是由该符号连接引用文件的信息。

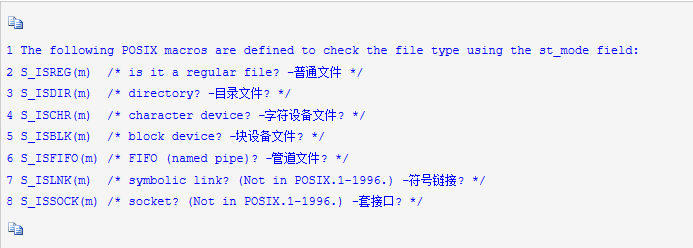
struct stat在系统头文件<stat.h>中，具体定义如下：



st\_mode是用特征位来表示文件类型以及操作权限的，特征位的定义如下：



POSIX定义了下面几种通过st\_mode判断文件类型的宏：



**11.verlog中的timescale用法**

verilog-timescale解释

`timescale 1ns/1ps

其中1ns代表步进，1ps代表精度，精度代表小数点后能够识别到的位数

#200.001 //代表200ns\_1ps

#200.0001 //编译将会报错

描述:

timescale是Verilog HDL 中的一种时间尺度预编译指令，它用来定义模块的仿真 时的时间单位和时间精度。格式如下：

`timescale 仿真时间单位/时间精度

注意：用于说明仿真时间单位和时间精度的 数字只能是1、10、100，不能为其它的数字。而且，时间精度不能比时间单位还要大。最多两则一样大。比如：下面定义都是对的：

`timescale 1ns/1ps

`timescale 100ns/100ns

下面的定义是错的：

`timescale 1ps/1ns

时间精度就是模块仿真时间和延时的精确程序，比如：定义时间精度为10ns， 那么时序中所有的延时至多能精确到10ns，而8ns或者18ns是不可能做到的。

在编译过程中,timescale指令影响这一编译器指令后面所有模块中的时延值，直至遇到另一个timescale指令resetall指令。

在verilog中是没有默认timescale的，一个没有指定timescale的verilog模块就有可能错误的继承了前面编译模块的无效timescale参数.

下面举个简单的例子说明一下:

---------------------

原文：https://blog.csdn.net/super\_haifeng/article/details/50110371

12. 结构体中使用“.成员”=xxx方式初始化

linux内核结构体初始化时出现的.owner = THIS\_MODULE是C99新增的规定。叫指定初始化(**designated initializer**),我们看一个例子：

下面我们看一个例子，Linux-2.6.x/drivers/usb/storage/usb.c中有这样一个结构体初始化 项目：

static struct usb\_driver usb\_storage\_driver = {

.owner = THIS\_MODULE,

.name = \"usb-storage\",

.probe = storage\_probe,

.disconnect = storage\_disconnect,

.id\_table = storage\_usb\_ids, };

使用指定初始化，可以按任意顺序对结构体初始化，如：

struct book gitf={

.value =25.99,

.author = \”james Borne\”,

.title = \”Rue for the Toad\”

};

---------------------

原文：https://blog.csdn.net/bonnshore/article/details/7850382