昵称：azraelly
园龄：3年9个月
粉丝：42
关注：14
+加关注

< 2012年12月 >

日	一	二	三	四	五	六
25	26	27	28	29	30	1
2	3	4	5	6	7	8
9	10	11	12	13	14	15
16	17	18	19	20	21	22
23	24	25	26	27	28	29
30	31	1	2	3	4	5

搜索

找找看
谷歌搜索

常用链接

我的随笔
我的评论
我的参与
最新评论
我的标签
更多链接

随笔分类

D3D(1)
DDraw(3)
ffmpeg(7)
Linux编程(4)
编程问题(6)
网络编程(10)

随笔档案

2013年4月 (1)
2013年2月 (1)
2013年1月 (14)
2012年12月 (12)
2012年8月 (5)
2012年7月 (3)
2012年6月 (9)
2012年5月 (3)
2012年4月 (1)

最新评论

1. Re:live555
，但两者都未交代转载
--校园
2. Re:GCC 编译详解
求教了，对有些问题理解掌握的更好了
--东方无悔
3. Re:GCC 编译详解
写的认真，“使用aclocal生成一个“aclocal.m4”文件”，这个怎么生成不成功。生成一个autom4te.cache?
--白饭
4. Re:GCC 编译详解
@Thomas2015如果是生成的是汇编代码，是-S而不是-c啊...
--froge
5. Re:GCC 编译详解
@fengjieji博主对的好不，先编译再汇编

makefile文件编写详解

```
    echo begin delete ...
       rm -f *.o


   2.makefile自动化变量makefile

.PHONY:clean

#显式指定clean 为伪目标，防止在当前目录下存在clean文件是无法执行清理工作

OBJECTS=ping.o main.o

ping:$(OBJECTS)                                    #自定义变量

  g++ -Wall -g  $^ -o $@ -lpthread       #$^ 依赖列表的所有项ping.o main.o  $@ 表示目标项ping

ping.o:ping.cpp ping.h

   g++ -Wall -g -c $< -o $@                    #$<依赖列表的第一项ping.cpp

main.o:main.cpp

  @g++ -Wall -g -c $< -o $@                     #在命令前加@，表示不显示命令

clean:          rm -f $(OBJECTS)

#自动化变量和自定义变量的使用

makefile的执行：make clean -f makefile.bak   -f 指定要执行的makefile文件



   3.makefile默认推导规则

实例：

生成多个可执行文件

01Test.cpp 02Test.cpp

.PHONY:clean all

BIN=01Test 02Test

all:$(BIN)

clean:

   rm -f $(BIN)

执行后生成01Test 02Test可执行文件，系统执行的事隐含推导规则，也可以自己编写推导规则

模式规则：%.o:%.c

01Test.cpp 02Test.cpp

.PHONY:clean all
BIN=01Test 02Test
all:$(BIN)
%.o:%.c
       g++ -Wall -g -c $< -o $@
01Test:01Test.o
        g++ -Wall -g $^ -o $@
02Test:02Test.o
        g++ -Wall -g $^ -o $@
```

```
clean:
        rm -f *.o $(BIN)
```

后缀规则：.c.o:

```
.PHONY:clean all
BIN=01Test 02Test
all:$(BIN)
.c.o:
       g++ -Wall -g -c $< -o $@
01Test:01Test.o
        g++ -Wall -g $^ -o $@
02Test:02Test.o
        g++ -Wall -g $^ -o $@
clean:
        rm -f *.o $(BIN)
```

定义变量进一步简化makefile:

```
.PHONY:clean all

CC=g++

CFLAGS=-Wall -g
BIN=01Test 02Test
all:$(BIN)
.c.o:
       $(CC)  $(CFLAGS) -c $< -o $@
01Test:01Test.o
        $(CC) $(CFLAGS) $^ -o $@
02Test:02Test.o
         $(CC) $(CFLAGS) $^ -o $@
clean:
        rm -f *.o $(BIN)
```

多级目录makefile的实现:
1.make常用内嵌函数
2.二级目录的实现
实例

```
.PHONY:clean

 #显式指定clean 为伪目标，防止在当前目录下存在clean文件是无法执行清理工作

CC    =g++ CFLAGS =-Wall -g

BIN = ping SUBDIR =$(shell ls -d */)

ROOTSRC =$(wildcard *.cpp)    #当前目录下匹配模式的文件

ROOTOBJ =$(ROOTSRC:%.cpp=%.o)

SUBSRC =$(shell find $(SUBDIR) -name '*.cpp')  #指定目录下的源文件

SUBOBJ =$(SUBSRC:%.cpp=%.o)     #模式替换函数

$(BIN):$(ROOTOBJ) $(SUBOBJ)

     $(CC) $(CFLAGS) -o $(BIN) $(ROOTOBJ) $(SUBOBJ) -lpthread

.c.o:

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

clean:

     @rm -f $(BIN) $(ROOTOBJ) $(SUBOBJ)

#二级目录makefile文件的使用

          3.多级目录、多个makefile的实现
```

```
SUBDIRS = Test1 Test2
.PHONY:default all clean $(SUBDIRS)
default:all
all clean:
 $(MAKE) $(SUBDIRS) TARGET=$@
$(SUBDIRS):
 $(MAKE) -C $@ $(TARGET)
 #-C表示进入到目录 make -C Test1 all等价于make all Test1/makefile

Test1目录makefile
CC = g++
BIN = Test1
OBJS = Test1.o
.PHONY: all clean print
all:print $(BIN)
print:
 @echo "-----------make all in $(PWD) -----------"
$(BIN):$(OBJS)
 $(CC) $(OBJS) -o $@
%.o:%.cpp
 $(CC) -c $<
clean:
 @echo "-----------make clean in $(PWD) -----------"
 rm -f $(BIN) $(OBJS)
Test2目录的makefile
CC = g++
BIN = Test2
OBJS = Test2.o
CFLAGS = -Wall -g
.PHONY: all clean print
all:print $(BIN)
print:
 @echo "-----------make all in $(PWD) -----------"
$(BIN):$(OBJS)
 $(CC) $(CFLAGS) $(OBJS) -o $@
%.o:%.cpp
 $(CC) $(CFLAGS) -c $<
clean:
 @echo "-----------make clean in $(PWD) -----------"
 rm -f $(BIN) $(OBJS)
```

Linux 库的使用:

1.静态库（*.a）实质就是把.o文件的集合，归档打包

1.1生成.o文件: g++ -Wall hello_fn.c -o hello_fn.o

1.2生成.a文件: ar rcs libhello.a hello_fn.o (ar 是gnu归档工具，rcs('r' 静态库存在则替换replace，'c' 不存在则创建，'s' 保存.o文件的索引信息到库文件中去)

1.3 库文件的使用: g++ -Wall main.c libhello.a -o main 或者 g++ -Wall -L. main.c -o main

-lhello 由于gcc编译器默认不会搜索当前目录，索引用 -L. 在当前目录搜索库文件。

2.动态库(*.so or *.sa)

2.1生成.o文件: g++ -c Test.cpp

2.2生成.so文件: g++ -shared -fPIC Test.o -o libTest.so

2.3使用动态库: g++ -L. main.o - o main lTest. 当动态库和静态库同时存在是，优先使用动态库。

2.4程序运行动态库加载路径。 a. 拷贝.so文件到系统共享库/usr/lib/路径下。 b. 修改系统环境变量，增加库的路径。vi~/bash_profile LD_LIBRARY_PATH=库的路径（当前用户配置）。 c. ldconfig 配置ld.so.conf,ldconfig更新ld.so.cache vi /etc/ld.so.conf配置库文件路径（全局配置）

2.5 lld查看文件包含了哪些库文件。

3.库搜索路径:

3.1 C_INCLUDE_PATH、LIBRARY_PATH

3.2 -I 指定头文件目录、-L指定库文件搜索目录

3.3 环境变量指定的目录。 vi ~/.bash_profile export C_INCLUDE_PATH = /opt/soft/include/

export LIBRARY_PATH=/opt/soft/lib/ .~/.bash_profile 执行一下生效。 ~/ 当前用户根目录 /系统根目录

3.4系统指定的目录。一般系统默认的库头文件和库文件默认在 /usr/include/ /usr/lib/

分类: Linux编程

« 上一篇: GDB调试从基础到精通实例
» 下一篇: Windbg调试程序

posted @ 2012-12-23 17:39 azraelly 阅读(4082) 评论(0) 编辑 收藏