**Makefile整理**

1. Makefile基本概念
2. 1）在makefile中的命令，必须要以【tab】键开始。
3. make指令会在当前目录下找名字叫”makefile”或者”Makefile”的文件。
4. 如果找到，它会找文件中的第一个目标文件（target）,并把这个文件作为 最终的目标文件。
5. 基本规则

Makefile里主要包含了五个东西：显式规则，隐晦规则，变量定义，文件指示和注释。

1. 显示规则。显示规则说明了，如何生成一个或多个的目标文件。这是由makefile的书写者明显指出，要生成的文件，文件的依赖性，生成的命令。

Target...: prereqisites ...

Command

Target:可以是一个目标文件，也可以是执行文件。还可以是一个标签（Label）.

Prereqisites: 要生成所需要的文件或是目标。

Command: make需要执行的命令。

1. 隐晦规则。由于我们的make有自动推到的功能，所以隐晦规则可以让我们比较粗糙地简略书写makefile,这是由make所支持的。
2. 变量的定义。在makefile中我们要定义一系列变量，变量都是字符串，这个有点像c语言中的宏，当makefile被执行时，其中的变量都会被扩展到相应的应用位置上。

Object=mian.o tclser.o

如：我们声明一个变量，叫object,objs，或者OBJ。用来表示obj文件，后面使用”$(object)” 来获取变量的值。

1. 文件的指示。其中包括了三个部分，一个是makefile中引用另一个makefile,就像c语言中的include一样；另一个是根据某些情况指定makfile中的有效部分，就像c语言中的预编译 #if一样；还有就是定义一个多行的命令。
2. 注释。Makefile中中行注释，和unix和shell脚本一样，其注释使用”#”字符。如果你要在你的makefile中使用”#”字符，可以用反斜杠进行转义。”\#”
3. Make的工作方式

GNU的make工作时的执行步骤如下：

1. 读入所有的makefile.
2. 读入被include的其他makefile.
3. 初始化文件中的变量.
4. 推导隐晦规则，并分析所有规则.
5. 为所有的目标文件创建依赖关系链.
6. 根据依赖关系，决定哪些目标要重新生成.
7. 执行生成命令。

1-5 步为第一个阶段，6-7为第二个阶段。第一个阶段中，如果定义的变量被使用了，那么，make就会把其展开在使用的位置。但make并不会完全马上展开，make使用的时拖延战术，如果变量出现在依赖关系的规则中，那么仅当这条依赖被决定要使用了，变量才会在其内部展开。

1. 特殊符号

2.1 通配符 “\*”, ”?” ,”[...]”

\* : 表示任意一个或多个字符。

？ : 表示任意一个字符。

[..] : [abcd] 表示a,b,c,d中任意一个字符，[^abcd]表示除a,b,c,d以外的字符， [0-9]表示0~9中任意一个数字。

2.2 一般特殊符号

$@: 代表规则中的目标文件名。如果目标是一个文档（一般称.a文件为文档），那么它代表这个文档的文件名。在多目标的模式规则中，它代表的是那个触发规则被执行的目标文件名。

$%: 规则的目标文件是一个静态文件时，代表静态库的一个成员名。

$<: 规则的第一个依赖文件名。如果时隐含规则，则它代表通过目标指定的第一个依赖文件。

$? :

$^ : 规则的所有依赖文件列表，使用空格分隔。如果目标时静态文件名，它所代表的只能是所欲库成员（.o文件）名。一个文件可重复的出现在目标的依赖中，变量$^只记录它的一次引用情况。$^会去掉重复的依赖文件。

$(@D): [代表目标文件的目录部分(去掉目录部分的最后一个下划线)。如果”$@”是”dir/test.o”,则”$(@)D”表示“dir”](mailto:代表目标文件的目录部分(去掉目录部分的最后一个下划线)。如果”$@”是”dir/test.o”,则”$(@)D”表示“dir”)

?= : 该符号左边的变量，如果在本条语句之前没有定义过，则执行本语句，如果已经定义，那么本语句什么都不做。

+= : 是添加等号后面的值。

CC : c语言编译器，默认值为gcc

CFLAGS: c语言编译器的编译选项。

2.3 文件搜寻 VPATH

1）一般搜索（变量 VPATH）

GNU make可以识别一个特殊变量”VPATH”。通过VPATH可以指定依赖文件的搜索路径，当规则的依赖文件在当前目录下不存在时，make会在此变量所指定的目录下去寻找这些依赖文件。通常我们都是用此变量来指定规则的依赖文件的搜索路径。其实”VPATH”变量所指定的时Makefile中所有文件的搜索路径，包括了规则的依赖文件和目标文件。

定义”VPATH”时使用空格或则冒号（:）将多个需要搜索的目录分开。Make搜索目录的顺序时按照变量”VPATH”定义中的目录顺序进行的（当前目录永远时第一搜索目录）。

2）选择性搜索（vpath）

2.4 伪目标

伪目标时这样一个目标：它不代表一个真正的文件名，在执行make时可以指定这个目标来执行其所在规则定义的命令，有时也可以将一个伪目标称为标签。（如 make all ,make clean , make a.out）使用伪目标有点原因：

1. 比如我们的makefile中定义只执行命令的目标和工作目录下实际文件出现名字冲突。（如目录下有clean这个文件，clean： 伪目标就会失效，需添加.PHONY:clean---明确的叫该目标声明成伪目标）
2. 提高执行make时的效率，特别时对于一个大型工程来说，编译的效率也许你同样关心。
3. 有一些目标，它的规则所定义的命令不是去创建文件，而仅仅通过make指定目标来执行一些特定系统命令或执行其依赖的目标规则。如all,也是伪目标的一种。

all: 执行主要的编译工作，而可以写多个目标文件。后面会一次执行这些目 标文件。

Install: 执行编译后的安装工作，把可执行文件，配置文件，文档等分别拷到 不同的安装目录。

clean: 删除编译后的二进制文件。

distclean: 删除除源文件之外的所有中间文件，如配置文件，文档等。

tags: 为vim等编译器生成tags文件。

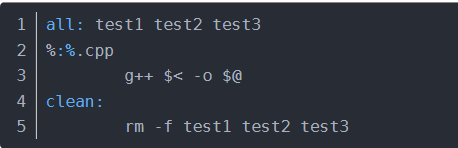
help : 打印当前makefile的帮助信息，比如有哪些目标可以有make指定去 执行。

·make处理makefile时，首先读取所有规则，建立关系依赖图。然后从缺省目标（第一个目标） 或者指定的目标开始执行。像clean,tags,这样的目标一般不会作为缺省目标，也不会跟缺省目标有任何依赖关系，所以make无法生成它的依赖关系和决定它是否要执行。所以要执行这样的目标时，必须要显示的指定make该目标。（如要清除中间文件，需执行make clean）。

1. 常用模式
2. %.o:%.c

所有的.o文件依赖于对应的.c文件

2) %:%.cpp

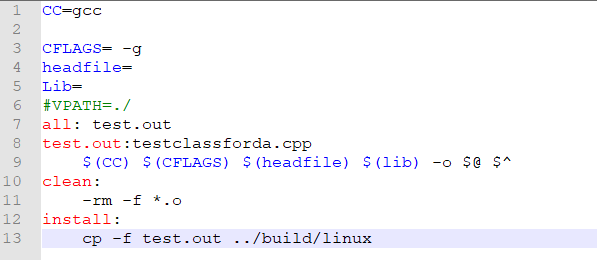


1. 包含多层makefile
2. make -C $(sub\_dir)

-C(大写c): 在子目录执行make.

1. 举例

1. 简单的单文件makefile



1. 第7行定义伪目标all,依赖项为test.out。于是文件就把这个依赖项当成了目标文件去生成。
2. 第12行伪目标install ，一般时编译以后安装的，这里做一个拷贝。



五 注意事项：

1. makefile 注意（链接顺序问题）

在写g++ lib -o a.out a.cpp 时如果a.cpp应用了lib中的库，可能会编译失败。

1. 对于library的查找，查找需要连接的符号名时从前向后找，根据-L指定的路径顺序查找，不同路径下的同名库，只取第一个（从左到右）,后面同名库被忽略。

在编译a.cpp的时候，扫描到lib ,没有发现被需要，但是在扫描到a.cpp的时候发现需要某个符号，会去右边找，这个时候会出现找不到的问题。所以依赖的库尽量放在a.cpp或者a.o的右边。这种情况编译器会认为不需要该库。

2.再使用时，如果加载了两个不同库的相同函数，那么调用的时候有可能会混响，添加这两个编译参数，强制加载当前库里的那个符号。

-Wl,-Bsymbolic.

其中Wl表示将紧跟其后的参数，传递给连接器ld。Bsymbolic表示强制采用本地的全局变量定义，这样就不会出现动态链接库的全局变量定义被应用程序/动态链接库中的同名定义给覆盖了。

六 编译完成以后的整理

1. 拷贝到output文件夹

outputdir: output

if [ ! -d $(OUTPUT\_PATH) ]; then mkdir -p $(OUTPUT\_PATH); fi

if [ ! -d $(OUTPUT\_PATH)/conf ]; then mkdir -p $(OUTPUT\_PATH)/conf; fi

if [ ! -d $(OUTPUT\_PATH)/dtmf ]; then mkdir -p $(OUTPUT\_PATH)/dtmf; fi

if [ ! -d $(OUTPUT\_PATH)/imsclient ]; then mkdir -p $(OUTPUT\_PATH)/imsclient; fi

cp conf/\* $(OUTPUT\_PATH)/conf

cp bgcc.cfg $(OUTPUT\_PATH)/

cp dtmf/8k-alaw/\* $(OUTPUT\_PATH)/dtmf

cp $(BINNAME) $(OUTPUT\_PATH)/

1. 删除

clean:

rm -f $(OBJS)

rm -f \*.a

rm -f $(BINNAME)

rm -fr $(OUTPUT\_PATH)

示例：

BINNAME = ims\_server

WORKROOT=../../

THIRD\_PATH=$(WORKROOT)/dep/

INC\_PATH = -I.

INC\_PATH += -I$(WORKROOT)/interface/output/bgcc/include/

INC\_PATH += -I$(WORKROOT)/interface/output/esl/include

INC\_PATH += -Iinc/fs

INC\_PATH += -Iinc/core

INC\_PATH += -Iinc/utl

INC\_PATH += -Iinc/api

INC\_PATH += -I$(WORKROOT)/interface/output/ims/include/

INC\_PATH += -I$(THIRD\_PATH)/boost/output/include

INC\_PATH += -I$(THIRD\_PATH)/libxml2/output/include

INC\_PATH += -I$(THIRD\_PATH)/pcre/output/include

INC\_PATH += -I$(THIRD\_PATH)/freeswitch/output/libs/esl/include

LIB\_PATH += -L$(THIRD\_PATH)/boost/output/lib

LIB\_PATH += -L$(THIRD\_PATH)/pcre/output/lib

LIB\_PATH += -L$(THIRD\_PATH)/libxml2/output/lib

LIB\_PATH += -L$(WORKROOT)/interface/output/esl/lib

LIB\_PATH += -L$(WORKROOT)/interface/output/ims/lib/linux/

LIB\_PATH += -L$(WORKROOT)/interface/output/bgcc/lib/

VERSION="1.2.0.4"

CXXFLAGS = -Wall -W -pipe -Wno-unused-parameter -Wreorder -Wwrite-strings -Wswitch -Wno-sign-compare\

-g -D\_\_XDEBUG\_\_ -D\_\_YDEBUG\_\_ \

-fPIC -finline-functions -Winline -Wreturn-type -Wtrigraphs -Wformat -Wparentheses -Wpointer-arith

-DVERSION=\"$(VERSION)\" -DCVSTAG=\"$(VERSION)\" -DPROJECT\_NAME=\"$(BINNAME)\"

LDFLAGS = -lboost\_system -lboost\_filesystem -lims -lbgcc -lssl -lesl -lxml2 -lpcre -lpthread -lcrypto -lrt -lm -lz -ldl -luuid

SOURCES = $(wildcard \*.cpp) $(wildcard src/fs/\*.cpp) $(wildcard src/utl/\*.cpp) $(wildcard src/core/\*.cpp) $(wildcard src/api/\*.cpp)

OBJS = $(patsubst %.cpp,%.o,$(SOURCES))

CXX = g++

OUTPUT\_PATH=./output/

#-----------------------------------------------------------------#

# make all

#-----------------------------------------------------------------#

all: outputdir

outputdir: output

if [ ! -d $(OUTPUT\_PATH) ]; then mkdir -p $(OUTPUT\_PATH); fi

if [ ! -d $(OUTPUT\_PATH)/conf ]; then mkdir -p $(OUTPUT\_PATH)/conf; fi

if [ ! -d $(OUTPUT\_PATH)/dtmf ]; then mkdir -p $(OUTPUT\_PATH)/dtmf; fi

if [ ! -d $(OUTPUT\_PATH)/imsclient ]; then mkdir -p $(OUTPUT\_PATH)/imsclient; fi

cp conf/\* $(OUTPUT\_PATH)/conf

cp bgcc.cfg $(OUTPUT\_PATH)/

cp dtmf/8k-alaw/\* $(OUTPUT\_PATH)/dtmf

cp $(BINNAME) $(OUTPUT\_PATH)/

test: output

ar -r libims\_server.a $(OBJS)

output: $(OBJS)

$(CXX) $(CXXFLAGS) -o $(BINNAME) $(OBJS) $(INC\_PATH) $(LIB\_PATH) $(LDFLAGS)

$(OBJS): %.o: %.cpp

$(CXX) $(CXXFLAGS) -o $@ $(INC\_PATH) -c $<

clean:

rm -f $(OBJS)

rm -f \*.a

rm -f $(BINNAME)

rm -fr $(OUTPUT\_PATH)