# Linux 工程管理工具
# make入门

Version 1.0

西安電子科技大學

## 需要掌握的要点

- 理解make工具的工作原理
- 掌握makefile文件显示规则的写法
- 理解make 变量的用法
- 理解make隐式规则
- 能够读懂简单的makefile文件



西安電子科技大學

# make工具的功能

- 主要负责一个软件工程中多个源代码的自动编译工作
- 还能进行环境检测、后期处理等工作；
- make工具可以识别出工程中哪些文件已经被修改，并且在再次编译的时候只编译这些文件，从而提高编译的效率；
- make的主要任务是根据makefile文件（一个脚本文件）中定义的规则和步骤，根据各个模块的更新情况，自动完成整个软件项目的维护和目标程序生成工作。让程序员把注意力放在“代码本身”，而不是“编译代码”上。



# makefile文件

- makefile告诉make该做什么、怎么做
- make工具在当前目录下寻找名为“Makefile”或“makefile”的文件并解释执行其指令。
- makefile主要包含了
  1）一系列规则（显示规则、隐式规则、模式规则）
  2）变量定义
  3）文件指示（指示包含其他makefile文件，或根据情况指定makefile文件中的有效部分）

# Makefile的规则

- 规则

  **一条规则包含3个方面的内容，**

  1）要创建的目标（文件），

  2）创建目标（文件）所依赖的文件列表；

  3）通过依赖文件创建目标文件的命令组



# Makefile的规则

- **规则一般形式**

  *target: dependency_files*

  *<tab>command*

  <tab>...

- **例如**

  test:test.c

  gcc –O –o test test.c

- **Makefile 与 makefile**

  makefile优先

# 一个简单的makefile

```
edit : main.o kbd.o command.o display.o insert.o search.o files.o utils.o
        gcc -o edit main.o kbd.o command.o display.o insert.o \
            search.o files.o utils.o
main.o : main.c defs.h
        gcc -c main.c
kbd.o : kbd.c defs.h command.h
        gcc -c kbd.c
command.o : command.c defs.h command.h
        gcc -c command.c
display.o : display.c defs.h buffer.h
        gcc -c display.c
insert.o : insert.c defs.h buffer.h
        gcc -c insert.c
search.o : search.c defs.h buffer.h
        gcc -c search.c
files.o : files.c defs.h buffer.h command.h
        gcc -c files.c
utils.o : utils.c defs.h
        gcc -c utils.c

clean :
    rm edit main.o kbd.o command.o display.o insert.o \
            search.o files.o utils.o
```



# 另一个简单的makefile

```
all     : test

test    : calc.o main.o
          gcc –o test calc.o main.o –lm

main.o  : main.c calc.h
          gcc –c –Wall main.c

calc.o  : calc.c
          gcc –c –Wall calc.c

clean   :
          rm  *.o test
```

# Makefile 中的变量

- 变量可以指代一个长的字符串
- 使用变量可以
  - 降低错误风险
  - 简化makefile
- 例：objects变量（$(objects)）

```
objects = main.o kbd.o command.o \
          display.o insert.o search.o files.o utils.o
edit : $(objects)
          cc -o edit $(objects)
```



# Makefile 中的变量

- 简单变量　　VAR := var
- 递归变量　　VAR = var
- 自动变量

# Makefile 中的变量—预定义变量

| 命令格式 | 含　义 |
| --- | --- |
| AR | 库文件维护程序的名称，默认值为ar |
| AS | 汇编程序的名称，默认值为as |
| CC | C编译器的名称，默认值为cc |
| CPP | C预编译器的名称，默认值为$(CC) – E |
| CXX | C++编译器的名称，默认值为g++ |
| FC | FORTRAN编译器的名称，默认值为f77 |
| RM | 文件删除程序的名称，默认值为rm – f |
| ARFLAGS | 库文件维护程序的选项，无默认值 |
| ASFLAGS | 汇编程序的选项，无默认值 |
| CFLAGS | C编译器的选项，无默认值 |
| CPPFLAGS | C预编译的选项，无默认值 |
| CXXFLAGS | C++编译器的选项，无默认值 |
| FFLAGS | FORTRAN编译器的选项，无默认值 |



# Makefile 中的变量—常见自动变量

| 命令格式 | 含　义 |
| --- | --- |
| $* | 不包含扩展名的目标文件名称 |
| $+ | 所有的依赖文件，以空格分开，并以出现的先后为序，可能包含重复的依赖文件 |
| $< | 第一个依赖文件的名称 |
| $? | 所有时间戳比目标文件晚的依赖文件，并以空格分开 |
| $@ | 目标文件的完整名称 |
| $^ | 所有不重复的依赖文件，以空格分开 |
| $% | 如果目标是归档成员，则该变量表示目标的归档成员名称 |

# 又一个简单的makefile

```
OBJS = kang.o yul.o
CC = gcc
CFLAGS = -Wall -O -g
david : $(OBJS)
	$(CC) $(OBJS) -o david
kang.o : kang.c kang.h
	$(CC) $(CFLAGS) -c kang.c -o kang.o
yul.o : yul.c yul.h
	$(CC) $(CFLAGS) -c yul.c -o yul.o
```



# 又一个简单的makefile（自动变量）

```
OBJS = kang.o yul.o
CC = gcc
CFLAGS = -Wall -O -g
david : $(OBJS)
	$(CC) $^ -o $@
kang.o : kang.c kang.h
	$(CC) $(CFLAGS) -c $< -o $@
yul.o : yul.c yul.h
	$(CC) $(CFLAGS) -c $< -o $@
```

# Makefile 中的规则

- 显式规则
- 隐式规则
  - 由make工具自动推导由依赖文件生成目标文件的规则

Makefile中常见隐式规则目录

| 对应语言后缀名 | 规　则 |
|---|---|
| C编译：.c变为.o | $(CC) –c $(CPPFLAGS) $(CFLAGS) |
| C++编译：.cc或.C变为.o | $(CXX) -c $(CPPFLAGS) $(CXXFLAGS) |
| Pascal编译：.p变为.o | $(PC) -c $(PFLAGS) |
| Fortran编译：.r变为-o | $(FC) -c $(FFLAGS)。 |

一个.c 对应 一个 .h



# Makefile 中的规则

- 模式规则
  - 模式规则能引入用户自定义变量，为多个文件建立相同的规则（模板），由make工具根据模板自动推导由依赖文件生成目标文件的规则
  - 模式规则的格式类似于普通规则，这个规则中的相关文件前必须用“%”标明。

```
OBJS = kang.o yul.o
CC = gcc
CFLAGS = -Wall -O -g
david : $(OBJS)
    $(CC) $^ -o $@
%.o : %.c
    $(CC) $(CFLAGS) -c $< -o $@
```

# 又一个简单的makefile

```
CC = gcc
OBJECTS = calc.o main.o
LIBS = -lm
all     : test

test    : $(OBJECTS)
          $(CC) -o test $(OBJECTS) $(LIBS)

main.o : main.c calc.h
calc.o : calc.c

clean   :
          rm  *.o test
```





# 一个不太简单的makefile

```
1 #Debug information : 1= yes , 0 = no
2 DEBUG = 1
3 CFLAGS = -Wall
4 ifeq ($(DEBUG),1)
5 CFLAGS += -g
6 else
7 CFLAGS += -O2
8 endif
9
10 EXE = test
11 SRC = $(wildcard *.c)
12 CC = gcc
13 LIBS = -lm
14 OBJ = $(SRC:.c=.o)
15 OBJDIR = TES
16 DEPEND = .depend
```

.
- abc.c
- abc.h
- main.c
- makefile
- xyz.c
- xyz.h

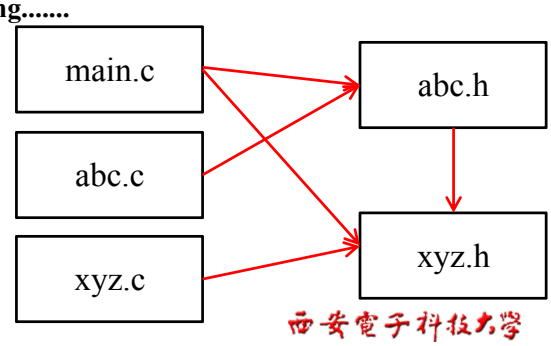
# 一个不太简单的makefile-续

```
17
18 $(EXE):$(OBJ)
19     $(CC) $(CFLAGS) -o $@ $^
20
21 $(DEPEND):
22 #   $(CC) -MM $(SRC) | > 1
23 #   @$(CC) -MM $(SRC) | sed 's/\(.*\)\(.o\)[:]/$(OBJDIR)\/\1.o:/g' \
24 #       > $(DEPEND)
25
26     @$(CC) -MM $(SRC) | sed 's/\(.*\)\(.o\)[:]/\1.o:/g' \
27     > $(DEPEND)
28
29     -include $(DEPEND)
30 clean:
31     @rm $(EXE) $(OBJ) $(DEPEND) -f
```



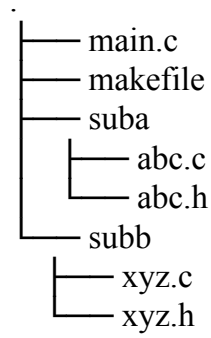
# 一个不太简单的makefile-续

```
$make -n
gcc -Wall -g   -c -o abc.o abc.c
gcc -Wall -g   -c -o main.o main.c
gcc -Wall -g   -c -o xyz.o xyz.c
gcc -Wall -g -o test abc.o main.o xyz.o
./test
abc sub runing.....
called by abc>>  xyz sub runing.....
xyz sub runing.....
```



# 不太简单的makefile—增强版

```
1 #Debug information: 1=yes,0=no
2 DEBUG = 1
3 CFLAGS = -Wall
4
5 ifeq ($(DEBUG),1)
6 CFLAGS += -g
7 else
8 CFLAGS += -O2
9 endif
10
11 EXE = test
12 CC = gcc
13 LIB = -lm
14 SRC = $(wildcard *.c)
15 SRC += $(wildcard ./suba/*.c)
16 SRC += $(wildcard ./subb/*.c)
17
18 OBJ = $(SRC:.c=.o)
19 DEP = .dep
```





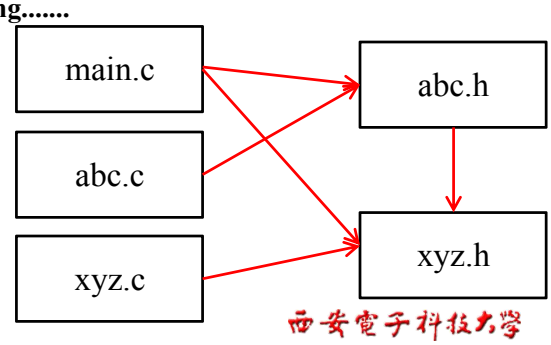

# 不太简单的makefile—增强版-续

```
20
21 $(EXE):$(OBJ)
22     $(CC) $(CFLAGS) -o $@ $^
23 $(DEP):
24     @$(CC) -MM $(SRC) | sed 's/\(.*\)\(.o\)[:]/\1.o:/g' > $(DEP)
25 -include $(DEP)
26
27 clean:
28     @rm $(EXE) $(OBJ) $(DEP) -f
```

# 不太简单的makefile--增强版-续

```
$make -n
gcc -Wall -g   -c -o main.o main.c
gcc -Wall -g   -c -o suba/abc.o suba/abc.c
gcc -Wall -g   -c -o subb/xyz.o subb/xyz.c
gcc -Wall -g -o test main.o suba/abc.o subb/xyz.o
./test
abc sub runing.....
called by abc>>  xyz sub runing.....
xyz sub runing.....
```

解决了跨目录文件编译问题






# make使用

| 命令格式 | 含　义 |
|---|---|
| -C dir | 读入指定目录下的Makefile |
| -f file | 读入当前目录下的file文件作为Makefile |
| -i | 忽略所有的命令执行错误 |
| -I dir | 指定被包含的Makefile所在目录 |
| -n | 只打印要执行的命令，但不执行这些命令 |
| -p | 显示make变量数据库和隐含规则 |
| -s | 在执行命令时不显示命令 |
| -w | 如果make在执行过程中改变目录，打印当前目录名 |

**http://blog.csdn.net/liang13664759/article/details/1771246/**