

# 第 00 章作业 - Linux 知识补充 - Makefile 文件的作用及编写方法

1652817 钟钰琛 计算机科学与技术

2018 年 10 月 10 日

## 1 Makefile 文件的作用

*makefile* 是用来告诉 *make* 做什么. 一般来讲, *makefile* 是用来编译代码和链接程序.

*makefile* 包含一套规则, 通过被 *make* 解释来产生目标.*makefile* 文件描述了整个工程所有文件的编译顺序、编译规则. 而且在 *makefile* 中, 可以使用系统 Shell 所提供的任何命令.

有了 *makefile*, 整个工程的编译, 只需要一个命令就可以完成“自动化编译”. 可以说, *makefile* 是构建和管理自己工程的一份说明书.

## 2 Makefile 文件的基本语法

*makefile* 中定义规则的语法:

---

```
TARGET...: PREREQUISITES...  
    COMMAND  
    ...  
    ...
```

---

**target:** 规则的目标, 可以是最后需要生成的文件也可以是中间过程文件名. 也可以是一个 *make* 执行动作的名称 (伪目标).

**prerequisites:** 规则的依赖. 为了生成规则目标, 所需要的文件名列表. 分两种, *normal* 和 *order-only*.

**command:** 规则的命令行. 是规则需要执行的动作. 每一个命令需要以 [TAB] 字符开始.

在上次执行 **make** 后, 如果修改了其依赖 (**normal** 类型) 的文件, 则需要根据规则重新生成; 如果尚不存在目标文件, 则按照规则生成; 否则将什么都不做.

如果规则的目标是一个动作的名称, 比如 **clean**, 那么执行这个动作只需要输入 **make clean** 就可以了.

*makefile* 指定变量: 可以添加变量来替代一些名字或者名字列表, 方便维护.

---

```
OBJ=main.o foo.o bar.o
```

---

然后可以用 **\$(OBJ)** 来表示它.

自动化变量:

- **\$@** 表示规则的目标文件名
- **\$<** 规则的第一个依赖文件名
- **\$^** 规则的所有依赖文件列表

另外, 在使用 **make** 编译.c 源文件时, 编译.c 源文件规则的命令可以不用明确给出. 这是因为 **make** 会自动为.o 文件寻找对应的依赖文件 (文件名除后缀外, 其余都相同). 但其余的依赖文件需要明确给出.

*makefile* 静态模式:

---

```
TARGET...: TARGET-PATTERN:PREREQ-PATTERNS...  
COMMANDS  
...
```

---

**targets:** 列出此规则的一系列目标文件

**target-pattern** 和 **prereq-pattern** 说明了如何为每一个目标文件生成依赖文件. 模式字符% 可以匹配目标文件中的任何部分, % 匹配的部分就是“茎”. 从目标模式的目标名字中抽取“茎”替代依赖模式中的相应部分来产生对应目标的依赖文件.

调用其他 *makefile* 用 **\$(MAKE)**

### 3 给出下列几种常用情况的 Makefile 文件的写法

#### 3.1 目录结构

```
[root@vm-linux 1652817-000103]# ll
总用量 0
drwxr-xr-x 2 root root 6 10月  9 22:28 01
drwxr-xr-x 2 root root 6 10月  9 22:28 02
drwxr-xr-x 2 root root 6 10月  9 22:28 03
[root@vm-linux 1652817-000103]#
```

图 1: 目录结构

#### 3.2 子目录 01

makefile 文件内容如下:

---

```
CC=gcc
OBJ=test1.o test2.o test3.o

test: $(OBJ)
    $(CC) -o $@ $^

clean:
    rm test *.o
```

---

1. 执行 make

```
192.168.159.22 x
[root@vm-linux 01]# make
gcc -c -o test1.o test1.c
gcc -c -o test2.o test2.c
gcc -c -o test3.o test3.c
gcc -o test test1.o test2.o test3.o
[root@vm-linux 01]# ll
总用量 40
-rw-r--r-- 1 root root 86 10月 9 23:19 makefile
-rwxr-xr-x 1 root root 8568 10月 9 23:21 test
-rw-r--r-- 1 root root 68 10月 9 23:11 test1.c
-rw-r--r-- 1 root root 1496 10月 9 23:21 test1.o
-rw-r--r-- 1 root root 67 10月 9 22:30 test2.c
-rw-r--r-- 1 root root 1496 10月 9 23:21 test2.o
-rw-r--r-- 1 root root 92 10月 9 22:30 test3.c
-rw-r--r-- 1 root root 1432 10月 9 23:21 test3.o
[root@vm-linux 01]#
```

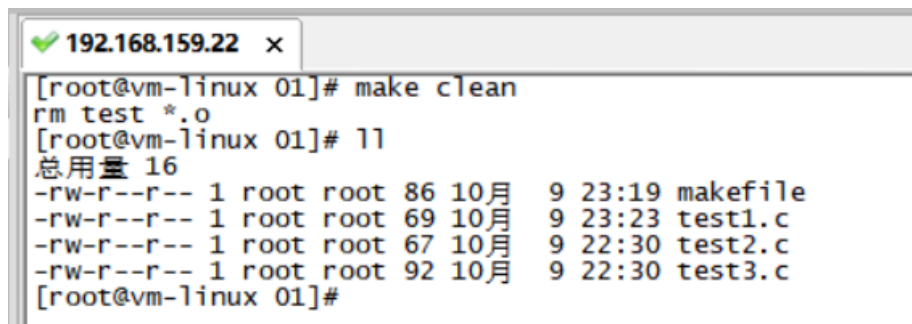
图 2: make

2. 任何一个.c 变化, 只编译这个改变的.c, 其余两个不需要重新编译.

```
"test1.c" 7L, 69C 已写入
[root@vm-linux 01]# ls
makefile test test1.c test1.o test2.c test2.o test3.c test3.o
[root@vm-linux 01]# ll
总用量 40
-rw-r--r-- 1 root root 86 10月 9 23:19 makefile
-rwxr-xr-x 1 root root 8568 10月 9 23:21 test
-rw-r--r-- 1 root root 69 10月 9 23:23 test1.c 更新的test1.c
-rw-r--r-- 1 root root 1496 10月 9 23:21 test1.o
-rw-r--r-- 1 root root 67 10月 9 22:30 test2.c
-rw-r--r-- 1 root root 1496 10月 9 23:21 test2.o
-rw-r--r-- 1 root root 92 10月 9 22:30 test3.c
-rw-r--r-- 1 root root 1432 10月 9 23:21 test3.o
[root@vm-linux 01]# make
gcc -c -o test1.o test1.c
gcc -o test test1.o test2.o test3.o
[root@vm-linux 01]# ll
总用量 40
-rw-r--r-- 1 root root 86 10月 9 23:19 makefile
-rwxr-xr-x 1 root root 8568 10月 9 23:24 test
-rw-r--r-- 1 root root 69 10月 9 23:23 test1.c
-rw-r--r-- 1 root root 1496 10月 9 23:24 test1.o 执行make后, .o中只有test1.o被更新
-rw-r--r-- 1 root root 67 10月 9 22:30 test2.c
-rw-r--r-- 1 root root 1496 10月 9 23:21 test2.o
-rw-r--r-- 1 root root 92 10月 9 22:30 test3.c
-rw-r--r-- 1 root root 1432 10月 9 23:21 test3.o
[root@vm-linux 01]#
```

图 3: make

### 3. 执行 make clean



```
[root@vm-linux 01]# make clean
rm test *.o
[root@vm-linux 01]# ll
总用量 16
-rw-r--r-- 1 root root 86 10月  9 23:19 makefile
-rw-r--r-- 1 root root 69 10月  9 23:23 test1.c
-rw-r--r-- 1 root root 67 10月  9 22:30 test2.c
-rw-r--r-- 1 root root 92 10月  9 22:30 test3.c
[root@vm-linux 01]#
```

图 4: make clean

## 3.3 子目录 02

makefile 文件内容如下:

---

```
CC=gcc
CFLAGS=-I.
DEPS=test.h
OBJ=test1.o test2.o test3.o

%.o: %.c $(DEPS)
    $(CC) -c -o $@ $< $(CFLAGS)

test: $(OBJ)
    $(CC) -o $@ $^ $(CFLAGS)

clean:
    rm test *.o
```

---

1. 执行 make
2. 改变 test.h 中的 a 值后, 再次 make, 会重新编译这个三个.c

```

192.168.159.22 x
[root@vm-linux 02]# make
gcc -c -o test1.o test1.c -I.
gcc -c -o test2.o test2.c -I.
gcc -c -o test3.o test3.c -I.
gcc -o test test1.o test2.o test3.o -I.
[root@vm-linux 02]# ll
总用量 44
-rw-r--r-- 1 root root 165 10月 9 23:39 makefile
-rwxr-xr-x 1 root root 8616 10月 9 23:39 test
-rw-r--r-- 1 root root 87 10月 9 23:30 test1.c
-rw-r--r-- 1 root root 1600 10月 9 23:39 test1.o
-rw-r--r-- 1 root root 86 10月 9 23:31 test2.c
-rw-r--r-- 1 root root 1568 10月 9 23:39 test2.o
-rw-r--r-- 1 root root 62 10月 9 23:31 test3.c
-rw-r--r-- 1 root root 1424 10月 9 23:39 test3.o
-rw-r--r-- 1 root root 56 10月 9 23:31 test.h
[root@vm-linux 02]#

```

图 5: make

```

~
"test.h" 4L, 58C 已写入
[root@vm-linux 02]# ls
makefile test test1.c test1.o test2.c test2.o test3.c test3.o test.h
[root@vm-linux 02]# ll
总用量 44
-rw-r--r-- 1 root root 165 10月 9 23:39 makefile
-rwxr-xr-x 1 root root 8616 10月 9 23:41 test
-rw-r--r-- 1 root root 87 10月 9 23:30 test1.c
-rw-r--r-- 1 root root 1600 10月 9 23:41 test1.o
-rw-r--r-- 1 root root 88 10月 9 23:41 test2.c
-rw-r--r-- 1 root root 1600 10月 9 23:41 test2.o
-rw-r--r-- 1 root root 62 10月 9 23:31 test3.c
-rw-r--r-- 1 root root 1424 10月 9 23:41 test3.o
-rw-r--r-- 1 root root 58 10月 9 23:41 test.h
[root@vm-linux 02]# make
gcc -c -o test1.o test1.c -I.
gcc -c -o test2.o test2.c -I.
gcc -c -o test3.o test3.c -I.
gcc -o test test1.o test2.o test3.o -I.
[root@vm-linux 02]# ll
总用量 44
-rw-r--r-- 1 root root 165 10月 9 23:39 makefile
-rwxr-xr-x 1 root root 8616 10月 9 23:42 test
-rw-r--r-- 1 root root 87 10月 9 23:30 test1.c
-rw-r--r-- 1 root root 1600 10月 9 23:42 test1.o
-rw-r--r-- 1 root root 88 10月 9 23:41 test2.c
-rw-r--r-- 1 root root 1600 10月 9 23:42 test2.o
-rw-r--r-- 1 root root 62 10月 9 23:31 test3.c
-rw-r--r-- 1 root root 1424 10月 9 23:42 test3.o
-rw-r--r-- 1 root root 58 10月 9 23:41 test.h
[root@vm-linux 02]#

```

修改test.h后，再次make  
test1.c test2.c  
test3.c  
都重新编译了

图 6: make

### 3. 执行 make clean

```
[root@vm-linux 02]# make clean
rm test *.o
[root@vm-linux 02]# ll
总用量 20
-rw-r--r-- 1 root root 165 10月 9 23:39 makefile
-rw-r--r-- 1 root root 87 10月 9 23:30 test1.c
-rw-r--r-- 1 root root 88 10月 9 23:41 test2.c
-rw-r--r-- 1 root root 62 10月 9 23:31 test3.c
-rw-r--r-- 1 root root 58 10月 9 23:41 test.h
[root@vm-linux 02]#
```

图 7: make clean

## 3.4 子目录 03

makefile 文件内容如下:

```
CC=gcc
TARGET=test1 test2 test3
all: $(TARGET)
$(TARGET): % : %.c
    $(CC) -o $@ $<
clean:
    rm test?
```

这里用了静态模式, % 可以匹配 target 中任意目标, 匹配的部分在依赖模式中可以找到对应的.c

比如匹配了 test1, 那么“茎”就是“test1”, 然后对应的依赖就是“test1.c”. 其余以此类推.

其中 all 是一个伪目标, 如果只输入 make, 那么 all 就会成为默认目标, 也就会生成所有 target. 但如果指定了 test1, 就会只执行 test1 的目标.

#### 1. make

```
192.168.159.22 x
[root@vm-linux 03]# make
gcc -o test1 test1.c
gcc -o test2 test2.c
gcc -o test3 test3.c
[root@vm-linux 03]# ll
总用量 52
-rw-r--r-- 1 root root 100 10月 10 00:38 makefile
-rwxr-xr-x 1 root root 8440 10月 10 00:49 test1
-rw-r--r-- 1 root root 71 10月 9 23:54 test1.c
-rwxr-xr-x 1 root root 8440 10月 10 00:49 test2
-rw-r--r-- 1 root root 66 10月 9 23:54 test2.c
-rwxr-xr-x 1 root root 8440 10月 10 00:49 test3
-rw-r--r-- 1 root root 74 10月 9 23:55 test3.c
[root@vm-linux 03]#
```

图 8: make

2. make test1 ...

```
192.168.159.22 x
[root@vm-linux 03]# ls
makefile test1.c test2.c test3.c
[root@vm-linux 03]# make test1
gcc -o test1 test1.c
[root@vm-linux 03]# ls
makefile test1 test1.c test2.c test3.c
[root@vm-linux 03]# make test2
gcc -o test2 test2.c
[root@vm-linux 03]# ls
makefile test1 test1.c test2 test2.c test3.c
[root@vm-linux 03]# make test3
gcc -o test3 test3.c
[root@vm-linux 03]# ls
makefile test1 test1.c test2 test2.c test3 test3.c
[root@vm-linux 03]# ./test1
1652817
[root@vm-linux 03]# ./test2
钟钰琛
[root@vm-linux 03]# ./test3
1652817 钟钰琛
[root@vm-linux 03]#
```

图 9: make

3. make clean 注意这里没有生成中间.o



```

[root@vm-linux 03]# make clean
rm test?
[root@vm-linux 03]# ll
总用量 16
-rw-r--r-- 1 root root 100 10月 10 00:38 makefile
-rw-r--r-- 1 root root 71 10月 9 23:54 test1.c
-rw-r--r-- 1 root root 66 10月 9 23:54 test2.c
-rw-r--r-- 1 root root 74 10月 9 23:55 test3.c
[root@vm-linux 03]#

```

图 10: make clean

### 3.5 总目录下

makefile 文件内容如下:

---

```

LISTDIR = `ls -d */`

subdirs:
    for dir in $(LISTDIR); \
    do \
        $(MAKE) -C $$dir; \
    done

clean:
    for dir in $(LISTDIR); \
    do \
        $(MAKE) clean -C $$dir; \
    done

```

---

这里是直接调用 shell 来找到所有文件夹

```
192.168.159.22 x
[root@vm-linux 1652817-000103]# ls
001 02 03 makefile
[root@vm-linux 1652817-000103]# make
for dir in `ls -d */`; \
do \
    make -C $dir; \
done
make[1]: 进入目录"/root/1652817-000103/001"
gcc -c -o test1.o test1.c
gcc -c -o test2.o test2.c
gcc -c -o test3.o test3.c
gcc -o test test1.o test2.o test3.o
make[1]: 离开目录"/root/1652817-000103/001"
make[1]: 进入目录"/root/1652817-000103/02"
gcc -c -o test1.o test1.c -I.
gcc -c -o test2.o test2.c -I.
gcc -c -o test3.o test3.c -I.
gcc -o test test1.o test2.o test3.o -I.
make[1]: 离开目录"/root/1652817-000103/02"
make[1]: 进入目录"/root/1652817-000103/03"
gcc -o test1 test1.c
gcc -o test2 test2.c
gcc -o test3 test3.c
make[1]: 离开目录"/root/1652817-000103/03"
[root@vm-linux 1652817-000103]# ls 001/
makefile test test1.c test1.o test2.c test2.o test3.c test3.o
[root@vm-linux 1652817-000103]# ls 02/
makefile test test1.c test1.o test2.c test2.o test3.c test3.o test.h
[root@vm-linux 1652817-000103]# ls 03/
makefile test1 test1.c test2 test2.c test3 test3.c
[root@vm-linux 1652817-000103]#
```

图 11: make

```
192.168.159.22 x
[root@vm-linux 1652817-000103]# make clean
for dir in `ls -d */`; \
do \
    make clean -C $dir; \
done
make[1]: 进入目录"/root/1652817-000103/001"
rm test *.o
make[1]: 离开目录"/root/1652817-000103/001"
make[1]: 进入目录"/root/1652817-000103/02"
rm test *.o
make[1]: 离开目录"/root/1652817-000103/02"
make[1]: 进入目录"/root/1652817-000103/03"
rm test?
make[1]: 离开目录"/root/1652817-000103/03"
[root@vm-linux 1652817-000103]# ls 001/
makefile test1.c test2.c test3.c
[root@vm-linux 1652817-000103]# ls 02/
makefile test1.c test2.c test3.c test.h
[root@vm-linux 1652817-000103]# ls 03/
makefile test1.c test2.c test3.c
[root@vm-linux 1652817-000103]#
```

图 12: make clean

默认情况下, make 会在工作目录寻找“GNUmakefile”、“makefile”和“Makefile”. 如果 makefile 重命名成其他名字, 那么需要在 make 指令中指定 -file=NAME