第十三章 Makefile

1: 什么是 make

make 是个命令，是个可执行程序，用来解析 Makefile 文件的命令

这个命令存放在 /usr/bin/

2：什么是 makefile?

makefile 是个文件，这个文件中描述了咱们程序的编译规则

咱们执行 make 命令的时候，make 命令会在当前目录下找 makefile 文件，

根据 makefile 文件里的规则，编译咱们的程序

注意：Makefile 规则文件是咱们程序员根据自己的程序，编写的编译规则

3：采用 Makefile 的好处

1、简化编译程序的时候输入得命令，编译的时候只需要敲 make 命令就可以了

2、可以节省编译时间，提高编译效率

1.1 make 概述

1、GNU make 是一种代码维护工具

2、make 工具会根据 makefile 文件定义的规则和步骤，完成整个软件项目的代码维护工作。

3、一般用来简化编译工作，可以极大地提高软件开发的效率。

4、windows 下一般由集成开发环境自动生成

5、linux 下需要由我们按照其语法自己编写

make 主要解决两个问题：

一、大量代码的关系维护

大项目中源代码比较多，手工维护、编译时间长而且编译命令复杂，难以记忆及维护

把代码维护命令及编译命令写在 makefile 文件中，然后再用 make 工具解析此文件自动执行相应命令，可实现代码的合理编译

二、减少重复编译时间

在改动其中一个文件的时候，能判断哪些文件被修改过，可以只对该文件进行重新编译，然后重新链接所有的目标文件，节省编译时间

1.2 makefile 语法及其执行

1.2.1 makefile 语法规则

目标：依赖文件列表

<Tab>命令列表

1、目标：

通常是要产生的文件名称,目标可以是可执行文件或其它 obj 文件,也可是一个动作的名称

2、依赖文件：

是用来输入从而产生目标的文件

一个目标通常有几个依赖文件（可以没有）

3、命令：

make 执行的动作,一个规则可以含几个命令（可以没有）

有多个命令时，每个命令占一行

例 1：简单的 Makefile 实例

main.c

```c
#include <stdio.h>
#include "main.h"
int main (void)
{
    printf("hello make world\n");
    printf("PI=%lf\n",PI);
    return 0;
}
```

main.h

```c
#define PI 3.1415926
```

makefile:

```
main:main.c main.h
    gcc main.c -o main
clean:
    rm main
```

1.2.2 make 命令格式

make [-f file] [targets]

1.[-f file]：

make 默认在工作目录中寻找名为 GNUmakefile、makefile、Makefile 的文件作为 makefile 输入文件

-f 可以指定以上名字以外的文件作为 makefile 输入文件

2.[targets]：

若使用 make 命令时没有指定目标，则 make 工具默认会实现 makefile 文件内的第一个目标，然后退出

指定了 make 工具要实现的目标，目标可以是一个或多个（多个目标间用空格隔开）。

例 2：稍复杂的 Makefile 实例

main.c 调用 printf1.c 中的 printf1 函数，同时需要使用 main.h 的中 PI，printf1.h 需要使用 main.h 中的 PI

main.c

```c
#include <stdio.h>
#include "main.h"
#include "printf1.h"
int main (void)
{
        printf("hello make world\n");
        printf("PI=%lf\n",PI);

        printf1();
        return 0;
}
```

printf1.c

```c
#include <stdio.h>
#include "main.h"
void printf1(void)
{
        printf("hello printf1 world PI=%lf\n",PI);
}
```

main.h

```c
#define PI 3.1415926
```

printf1.h

```c
extern void printf1();
```

稍微复杂的 makefile 编写

```
1 main:main.o printf1.o
2     gcc main.o printf1.o -o main
3 main.o:main.c main.h printf1.h
4     gcc -c main.c -o main.o
5 printf1.o:printf1.c main.h
6     gcc -c printf1.c -o printf1.o
7 clean:
8     rm *.o main
```

如 printf1.c 和 printf1.h 文件在最后一次编译到 printf1.o 目标文件后没有改动，它们不需重新编译
main 可以从源文件中重新编译并链接到没有改变的 printf1.o 目标文件。
如 printf1.c 和 printf1.h 源文件有改动，make 将在重新编译 main 之前自动重新编译 printf1.o。

1.2.3 假想目标:

前面 makefile 中出现的文件称之为假想目标
假想目标并不是一个真正的文件名，通常是一个目标集合或者动作
可以没有依赖或者命令
一般需要显示的使用 make + 名字 显示调用

```
all:exec1 exec2
clean:
<Tab>rm *.o exec
```

1.3 makefile 变量

1.3.1 makefile 变量概述

makefile 变量类似于 C 语言中的宏，当 makefile 被 make 工具解析时，其中的变量会被展开。
变量的作用：

保存文件名列表
保存文件目录列表
保存编译器名
保存编译参数
保存编译的输出
...

1.3.2 makefile 的变量分类：

1、自定义变量
在 makefile 文件中定义的变量。
make 工具传给 makefile 的变量。
2、系统环境变量
make 工具解析 makefile 前，读取系统环境变量并设置为 makefile 的变量。
3、预定义变量（自动变量）

1.3.3 自定义变量语法

定义变量：
变量名=变量值
引用变量：
$(变量名)或${变量名}
makefile 的变量名:
makefile 变量名可以以数字开头
注意：
1、变量是大小写敏感的
2、变量一般都在 makefile 的头部定义
3、变量几乎可在 makefile 的任何地方使用
例 2_2：
修改例 2 中的 makefile，使用自定义变量使其更加通用。

```
1  cc=gcc
2  #cc=arm-linux-gcc
3  obj=main.o printf1.o
4  target=main
5  cflags=-Wall -g
6
7  $(target):$(obj)
8      $(cc) $(obj) -o $(target) $(cflags)
9  main.o:main.c main.h printf1.h
10     $(cc) -c main.c -o main.o $(cflags)
11 printf1.o:printf1.c main.h
12     $(cc) -c printf1.c -o printf1.o $(cflags)
13 clean:
14     rm $(obj) $(target)
```

make 工具传给 makefile 的变量

执行 make 命令时，make 的参数 options 也可以给 makefile 设置变量。

#make cc=arm-linux-gcc

```
1 cc=gcc
2 main:main.c main.h
3     $(cc) main.c -o main
4 clean:
5     rm main
```

1.3.4 系统环境变量

make 工具会拷贝系统的环境变量并将其设置为 makefile 的变量，在 makefile 中可直接读取或修改拷贝后的变量。

#export test=10

#make clean

#echo $test

```
1 main:main.c main.h
2     gcc main.c -o main
3 clean:
4     rm main -rf
5     echo $(PWD)
6     echo "test=$(test)"
```

1.3.5 预定义变量

makefile 中有许多预定义变量，这些变量具有特殊的含义，可在 makefile 中直接使用。

$@ 目标名
$< 依赖文件列表中的第一个文件
$^ 依赖文件列表中除去重复文件的部分

AR 归档维护程序的程序名，默认值为 ar
ARFLAGS 归档维护程序的选项
AS 汇编程序的名称，默认值为 as
ASFLAGS 汇编程序的选项
CC C 编译器的名称，默认值为 cc
CFLAGS C 编译器的选项
CPP C 预编译器的名称，默认值为$(CC) -E
CPPFLAGS C 预编译的选项
CXX C++编译器的名称，默认值为 g++
CXXFLAGS C++编译器的选项

例 2_3：

修改例 2 中的 makefile，使用预定义变量,使其更加通用。

```
1  obj=main.o printf1.o
2  target=main
3  CFLAGS=-Wall -g
4
5  $(target):$(obj)
6      $(CC) $^ -o $@ $(CFLAGS)
7  main.o:main.c main.h printf1.h
8      $(CC) -c $< -o $@ $(CFLAGS)
9  printf1.o:printf1.c main.h
10     $(CC) -c $< -o $@ $(CFLAGS)
11 clean:
12     rm $(obj) $(target)
```