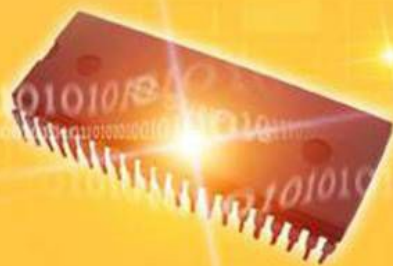


嵌入式系统工程师



Linux编程工具 (make)

- make概述
- makefile语法及其执行
- makefile变量

- make概述
- makefile语法及其执行
- makefile变量

make概述

GNU make是一种代码维护工具

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。

一般用来简化编译工作，可以极大地提高软件开发的效率。

windows下一般由集成开发环境自动生成

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

- make概述
- makefile语法及其执行
- makefile变量

➤ 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

```
1 #include <stdio.h>
2 #include "main.h"
3 int main (void)
4 {
5     printf("hello make world\n");
6     printf("PI=%lf\n", PI);
7     return 0;
8 }
```

main.h

```
1 #define PI 3.1415926
```

makefile:

```
1 main:main.c main.h
2     gcc main.c -o main
3 clean:
4     rm main
```

➤ 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

```
#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;
}
```

main.h

```
#define PI 3.1415926
```

printf1.c

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

printf1.h

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

➤ 例2: 稍复杂的Makefile实例

```
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。

➤ 假想目标:

前面makefile中出现的文件称之为假想目标

1. 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
2. 可以没有依赖或者命令
3. 一般需要显示的使用make + 名字 显示调用

```
all: exec1 exec2
```

```
clean:
```

```
<Tab>rm *.o exec
```

- make概述
- makefile语法及其执行
- makefile变量

- 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
 - ...

➤ makefile的变量分类:

➤ 自定义变量

在makefile文件中定义的变量。

make工具传给makefile的变量。

➤ 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。

➤ 预定义变量（自动变量）

➤ 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

➤ 变量是大小写敏感的

➤ 变量一般都在makefile的头部定义

➤ 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
```

➤ 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)"
```

➤ 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)
```

➤ Makefile练习

编写Makefile完成计算器程序(01_练习代码/calculator)的编译，并通过假想目标(clean)清除目标文件

- 第一步：不使用任何变量完成功能
- 第二步：使用自定义变量让程序更加通用
- 第三步：使用预定义变量让程序更加通用



凌阳教育官方微信：Sunplusedu

Tel: 400-705-9680 , BBS: www.51develop.net , QQ群: 241275518

