一、make概述
二、makefile语法及其执行
2.1 makefile语法规则
2.2 make命令格式
2.3 Makefile案例
sum.c
sub.c
head.h
main.c
Makefile
2.4 假想目标
三、makefile变量
3.1 makefile变量概述
3.2 变量的分类
3.3 自定义变量语法
3.4 系统环境变量
3.5 预定义变量

一、make概述

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、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

二、makefile语法及其执行

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
```

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）。

一般使用的时候直接make就可以

2.3 Makefile案例

sum.c

```c
#include "head.h"

int sum(int a, int b)
{
 return a + b;
}
```

sub.c

```c
#include "head.h"
```

```
2
3  int sub(int a, int b)
4  {
5   return a - b;
6  }
```

head.h

```
1  #ifndef _HEAD_H_
2  #define _HEAD_H_
3
4  #include <stdio.h>
5
6  int sum(int a, int b);
7  int sub(int a, int b);
8
9  #endif
```

main.c

```
1  #include "head.h"
2
3  int main(int argc, const char *argv[])
4  {
5   int x = 1000;
6   int y = 900;
7
8   printf("%d + %d = %d\n", x, y, sum(x, y));
9   printf("%d - %d = %d\n", x, y, sub(x, y));
10
11   return 0;
12  }
```

Makefile

```
1  main:main.o sub.o sum.o
2   gcc main.o sub.o sum.o -o main
3
4  main.o:main.c
5   gcc -c main.c -o main.o
6
7  sub.o:sub.c
8   gcc -c sub.c -o sub.o
9
10  sum.o:sum.c
11   gcc -c sum.c -o sum.o
```

```
12
13 clean:
14  rm *.o main a.out -rf
```

```
stu@qfedu:~/make$ make
gcc -c main.c -o main.o
gcc -c sub.c -o sub.o
gcc -c sum.c -o sum.o
gcc main.o sub.o sum.o -o main
stu@qfedu:~/make$ ls
a.out   main    main.o    Makefile1  Makefile3  sub.c  sum.c
head.h  main.c  Makefile  Makefile2  Makefile4  sub.o  sum.o
stu@qfedu:~/make$ ./main
1000 + 900 = 1900
1000 - 900 = 100
stu@qfedu:~/make$ make clean
rm *.o main a.out -rf
stu@qfedu:~/make$ ls
head.h  main.c  Makefile  Makefile1  Makefile2  Makefile3  Makefile4  sub.c  sum.c
stu@qfedu:~/make$
```

2.4 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

运行时使用make clean 就会执行clean后面的命令

三、makefile变量

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

3.2 变量的分类

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）

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}
注意：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

```
CC=gcc
obj=main
obj1=sub
obj2=sum
OBJ=main.o sub.o sum.o

$(obj):$(OBJ)
 $(CC) $(OBJ) -o $(obj)

$(obj).o:$(obj).c
 $(CC) -c $(obj).c -o $(obj).o

$(obj1).o:$(obj1).c
 $(CC) -c $(obj1).c -o $(obj1).o

$(obj2).o:$(obj2).c
 $(CC) -c $(obj2).c -o $(obj2).o

```

```
19 clean:
20  rm *.o $(obj) a.out -rf
```

执行结果

```
stu@qfedu:~/make$ make
gcc -c main.c -o main.o
gcc -c sub.c -o sub.o
gcc -c sum.c -o sum.o
gcc main.o sub.o sum.o -o main
stu@qfedu:~/make$ ./main
1000 + 900 = 1900
1000 - 900 = 100
```

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)"
```

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

```
CC=gcc
obj=main
obj1=sub
obj2=sum
OBJ=main.o sub.o sum.o
CFLAGS=-Wall -g

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

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

$(obj1).o:$(obj1).c
 $(CC) $(CFLAGS) -c $< -o $@

$(obj2).o:$(obj2).c
 $(CC) $(CFLAGS) -c $< -o $@

clean:
 rm *.o $(obj) a.out -rf
```

执行结果

```
stu@qfedu:~/make$ make
gcc -Wall -g -c main.c -o main.o
gcc -Wall -g -c sub.c -o sub.o
gcc -Wall -g -c sum.c -o sum.o
gcc main.o sub.o sum.o -o main
stu@qfedu:~/make$ ./main
1000 + 900 = 1900
1000 - 900 = 100
```

最精简版：

```
CC=gcc
obj=main
OBJ=main.o sub.o sum.o
CFLAGS=-Wall -g

```

```
$(obj):$(OBJ)
 $(CC) $^ -o $@

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

clean:
  rm *.o $(obj) a.out -rf
```