Makefile 系统学习

首先，有这么一个文件夹，里面的内容如下：

```
jack@jack-Ubuntu:~/LinuxCode/基础部分/Makefile$ ls
a.c  add.c  head.h  main.c  mul.c  src  sub.c
```

main.c 内容如下：

```c
#include <stdio.h>
#include "head.h"

int main(void)
{
    int a = 125;
    int sum = add(a, a);
    printf("========== =_= ==========\n");
    printf("%d + %d = %d\n", a, a, sum);
    printf("=========================\n");
    return 0;
}
```

现在编译运行这个程序，于是编写简单的 Makefile 如下：

```
app:main.c add.c sub.c mul.c
    gcc main.c add.c sub.c mul.c -o app
```

然后编译运行程序：

```
jack@jack-Ubuntu:~/LinuxCode/基础部分/Makefile$ make
gcc main.c add.c sub.c mul.c -o app
jack@jack-Ubuntu:~/LinuxCode/基础部分/Makefile$ ./app
这是一个加法运算!
========== =_= ==========
125 + 125 = 250
=========================
```

似乎一切完美！

但是缺点是，效率低，修改一个文件，全部文件需要重新编译。

于是，将 Makefile 修改如下：

```
app:main.o add.o sub.omul.o
    gcc main.o add.o sub.o mul.o -o app

main.o:main.c
    gcc main.c -c

add.o:add.c
    gcc add.c -c

sub.o:sub.c
    gcc sub.c -c
mul.o:mul.c
    gcc mul.c -c
```

```
app:main.o add.o sub.o mul.o
    gcc main.o add.o sub.o mul.o -o app

main.o:main.c
    gcc main.c -c

add.o:add.c
    gcc add.c -c

sub.o:sub.c
    gcc sub.c -c
mul.o:mul.c
    gcc mul.c -c
```

重新编译运行：

```
jack@jack-Ubuntu:~/LinuxCode/基础部分/Makefile$ make
gcc main.c -c
gcc add.c -c
gcc sub.c -c
gcc mul.c -c
gcc main.o add.o sub.o mul.o -o app
jack@jack-Ubuntu:~/LinuxCode/基础部分/Makefile$ ./app
这的确是一个加法运算!
========== =_= ==========
25 + 25 = 50
=========================
```

然后修改下 add.c 文件，再次编译运行：

```
jack@jack-Ubuntu:~/LinuxCode/基础部分/Makefile$ vi add.c
jack@jack-Ubuntu:~/LinuxCode/基础部分/Makefile$ make
gcc add.c -c
gcc main.o add.o sub.o mul.o -o app
jack@jack-Ubuntu:~/LinuxCode/基础部分/Makefile$ ./app
这是一个加法运算!
========== =_= ==========
25 + 25 = 50
=========================
jack@jack-Ubuntu:~/LinuxCode/基础部分/Makefile$
```

可见，只重新编译修改过的文件，其他的没有再次编译！

然而，这部分 Makefile 重复度太高，太多了，可以采用变量的方法来降低冗余。修改 Makefile 如下：

```
obj := main.o add.o sub.o mul.o
target := app
$(target):$(obj)
    gcc $(obj) -o $(target)
%.o:%.c
    gcc -c $< -o $@
```

```
obj := main.o add.o sub.o mul.o
target := app
$(target):$(obj)
    gcc $(obj) -o $(target)
%.o:%.c
    gcc -c $< -o $@
```

编译运行结果如下：

```
jack@jack-Ubuntu:~/LinuxCode/基础部分/Makefile$ ls
a.c  add.c  head.h  main.c  Makefile  mul.c  src  sub.c
jack@jack-Ubuntu:~/LinuxCode/基础部分/Makefile$ make
gcc -c main.c -o main.o
gcc -c add.c -o add.o
gcc -c sub.c -o sub.o
gcc -c mul.c -o mul.o
gcc main.o add.o sub.o mul.o -o app
jack@jack-Ubuntu:~/LinuxCode/基础部分/Makefile$ ./app
这是一个加法运算!
========== =_= ==========
25 + 25 = 50
=========================
```

现在解释下关于 Makefile 中的变量与规则：

- 自定义变量：

  obj=a.o b.o c.o

  obj=10

  变量的取值：aa=$(obj)

- makefile自带的变量：大写
  - CPPFLAGS
  - CC
- 自动变量：
  - $@: 规则中的目标
  - $<: 规则中的第一个依赖
  - $^: 规则中所有的依赖
  - 只能在规则的命令中使用

模式匹配：

```
%.o:%.c
```

第一次：main.o没有

```
main.o:main.c
    gcc -c main.c -o main.o
```

第二次：add.o

```
add.o:add.c
    gcc -c add.c -o add.o
```

当然，没有最简单，只有更简单，这样也是可以的：

```
4       gcc $^ -o $@
5       #gcc $(obj) -o $(target)
```

也许上面的也很完美了，但很不灵活，移植到其他项目中比较麻烦，于是引入了 Makefile 中的函数。

理论只是只有如下：

- makefile所有的函数都有返回值
- 查找指定目录下指定类型的文件
  - src = $(wildcard ./*.c)
- 匹配替换
  - obj = $(patsubst %.c, %.o, $(src))

修改 Makefile 如下：

```
src = $(wildcard ./*.c)
obj = $(patsubst %.c, %.o, $(src))
target = app
$(target):$(obj)
        gcc $^ -o $@
        #gcc $(obj) -o $(target)
%.o:%.c
        gcc -c $< -o $@
```

```
1 src = $(wildcard ./*.c)
2 obj = $(patsubst %.c, %.o, $(src))
3 target = app
4 $(target):$(obj)
5       gcc $^ -o $@
6 %.o:%.c
7       gcc -c $< -o $@
```

编译运行结果如下：

```
jack@jack-Ubuntu:~/LinuxCode/基础部分/Makefile$ ls
add.c  head.h  main.c  Makefile  mul.c  plus  sub.c
jack@jack-Ubuntu:~/LinuxCode/基础部分/Makefile$ make
gcc -c add.c -o add.o
gcc -c main.c -o main.o
gcc -c mul.c -o mul.o
gcc -c sub.c -o sub.o
gcc add.o main.o mul.o sub.o -o app
jack@jack-Ubuntu:~/LinuxCode/基础部分/Makefile$ ls
add.c  app      main.c  Makefile  mul.o  sub.c
add.o  head.h   main.o  mul.c     plus   sub.o
jack@jack-Ubuntu:~/LinuxCode/基础部分/Makefile$ ./app
这是一个加法运算!
========== =_= ==========
25 + 25 = 50
==========  ==========
```

Makefile 还需要清理工程，所以有：

- 让make生成不是终极目标的目标
  - make 目标名
- 编写一个清理项目的规则

  clean:

  rm *.o app
- 声明伪目标：

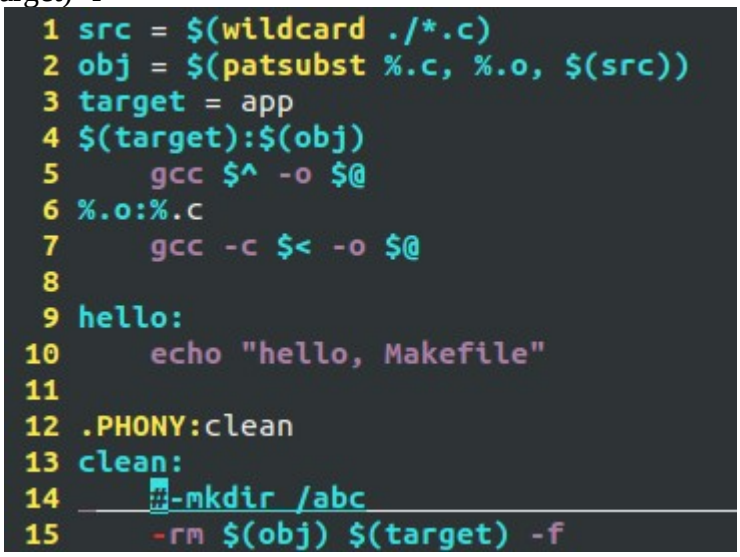
  .PHONY: clean
- -f: 强制执行
- 命令前加-
  - 忽略执行失败的命令，继续向下执行其余的命令

修改代码如下：

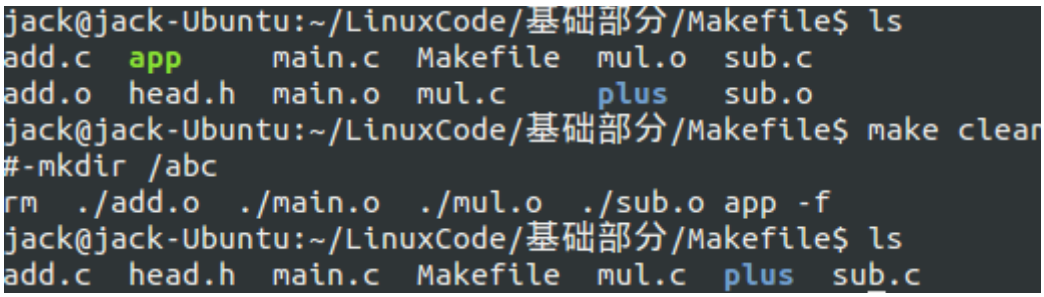
```
src = $(wildcard ./*.c)
obj = $(patsubst %.c, %.o, $(src))
target = app
$(target):$(obj)
	gcc $^ -o $@
%.o:%.c
	gcc -c $< -o $@

hello:
	echo "hello, Makefile"

.PHONY:clean
clean:
	-mkdir /abc
	-rm $(obj) $(target) -f
```



运行结果如下;



至此，Makefil 学习告一段落！