Makefile 学习

一、基础知识

1、什么是 makefile：

Linux 下，一个大型工程中有很多的源文件、头文件，它们有可能在不同的目录下，在开发过程中，每次修改都需要重新编译该项目，而编译命令可能会非常复杂，每次输入该命令很麻烦而且容易出错，此时若能够将该编译命令按一定的规则写到一个命名为 Makefile 或 makefile 的文件中，并且在每次编译时只需要一个简单的“make”命令即可完成编译，那么会为开发过程带来极大的便利。而且，大型项目每次编译都会消耗大量时间，如果只是个别文件修改，那么只需要重新编译这几个文件即可，makefile 会**智能**的选择编译那些修改过的文件，这样大大节省了项目开发时间。

2、编写规则：

1）三要素：目标、依赖项、命令

目标：最终要生成的目标文件。

依赖项：生成目标文件所需要的文件。

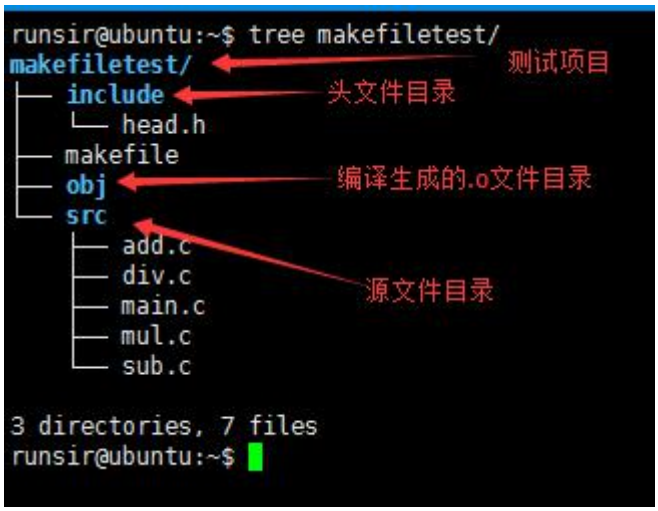
命令：执行的编译命令。命令前加上“-”号，如果这条命令出错则忽略这条命令继续向下执行，若不加“-”号，如果出错则直接退出。

2）格式：

目标:依赖条件

[Tab 缩进]命令

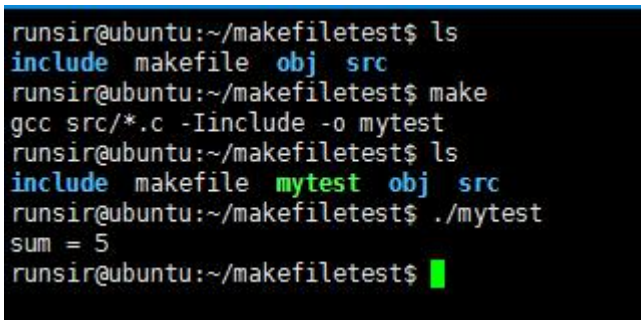
3、示例：



makefile 文件内容：

```
mytest:src/*.c
	gcc src/*.c -Iinclude -o mytest
```

执行结果：



4、原理：

输入 make 命令，自动寻找该目录下的 Makefile/makefile 文件，然后找到该文件中的第一个目标作为最终生成的目标文件，若该目标文件不存在或者所依赖文件的最后修改日期比目标文件更新，则执行命令。

注：直接输入 make 命令则默认执行第一条规则，输入 make + [目标] 命令则执行指定目标的规则。

二、makefile 优化

1、智能编译：

此时每次运行 make 所有的源文件都需要重新编译，消耗大量时间，可以作如下修改：

```
mytest:obj/main.o obj/add.o obj/sub.o obj/mul.o obj/div.o
    gcc obj/main.o obj/add.o obj/sub.o obj/mul.o obj/div.o -Iinclude -o mytest

obj/main.o:src/main.c
    gcc -c src/main.c -Iinclude -o obj/main.o

obj/add.o:src/add.c
    gcc -c src/add.c -o obj/add.o

obj/sub.o:src/sub.c
    gcc -c src/sub.c -o obj/sub.o

obj/mul.o:src/mul.c
    gcc -c src/mul.c -o obj/mul.o

obj/div.o:src/div.c
    gcc -c src/div.c -o obj/div.o
```

此时，输入 make 命令后，首先找到第一条规则，发现有依赖的.o 文件，于是向下寻找，找到能够产生依赖的.o 文件的规则并进行递归操作，直到第一条规则所依赖的.o 文件都是最新的，然后执行第一条规则中的命令。若开发中只修改了 add.c 文件，则 make 时只会重新编译 add.c 文件，并重新链接。

2、自定义变量：

obj/main.o obj/add.o obj/sub.o obj/mul.o obj/div.o 出现过多次，每次修改都需要修改多个地方很不方便且容易出错，所以可以用一个变量代替：

```
var=obj/main.o obj/add.o obj/sub.o obj/mul.o obj/div.o

mytest:$(var)
    gcc $(var) -Iinclude -o mytest
```

3、规则公式：

后四条规则很相似，可以用一个公式来代替（也可以认为是一个函数）。

规则公式中用 **%** 来匹配任意字符。

规则的命令中常用的自动变量有：

①$<：指代规则中的第一个依赖

②$^：指代规则中的所有依赖，每个依赖之间用空格隔开

③$@：指代规则中的目标

所以 makefile 可以修改为：

```
var=obj/main.o obj/add.o obj/sub.o obj/mul.o obj/div.o

mytest:$(var)
    gcc $(var) -Iinclude -o mytest

obj/main.o:src/main.c
    gcc -c src/main.c -Iinclude -o obj/main.o

obj/%.o:src/%.c
    gcc -c $< -o $@
```

4、伪目标：

伪目标就是这个规则中的目标并不是一个文件名，而是一个标签，最终不会生成目标文件，可以使用 **.PHONY:[目标]** 显示声明为伪目标。

每个 makefile 都应该有一个 clean 伪目标，用于删除生成的.o 文件和可执行文件，方便重新编译：

```
var=obj/main.o obj/add.o obj/sub.o obj/mul.o obj/div.o

mytest:$(var)
    gcc $(var) -Iinclude -o mytest

obj/main.o:src/main.c
    gcc -c src/main.c -Iinclude -o obj/main.o

obj/%.o:src/%.c
    gcc -c $< -o $@

.PHONY:clean
clean:
    rm $(var) mytest
```

三、makefile 中的函数

1、函数调用语法：

$(<function> <arguments>)
或
${<function> <arguments>}
以$开头，用小括号或者花括号括起来，<function>指代函数名，<arguments>指代参数，函数名和参数之间用空格隔开，参数之间用逗号隔开。
2、常用函数：
1）**wildcard 函数**：
用法：$(wildcard PATTERN...)
功能：返回指定目录下能够匹配 PATTERN 的所有文件名，多个文件名之间用空格隔开。
patsubst 函数：
用法：$(patsubst<pattern>,<replacement>,<text>)
功能：将<text>中的<pattern>替换为<replacement>。<pattern>可以包含通配符“%”，用来表示任意长度的字串。如果<replacement>中也包含“%”，那么<replacement>中的这个“%”将是<pattern>中的那个“%”所代表的字串。（可以用“\”来转义，以“\%”来表示真实含义的“%”字符）
2）makefile 修改：

```
src=$(wildcard src/*.c)
var=$(patsubst src/%.c,obj/%.o,$(src))

mytest:$(var)
	gcc $(var) -Iinclude -o mytest

obj/main.o:src/main.c
	gcc -c src/main.c -Iinclude -o obj/main.o

obj/%.o:src/%.c
	gcc -c $< -o $@

.PHONY:clean
clean:
	rm $(var) mytest
```