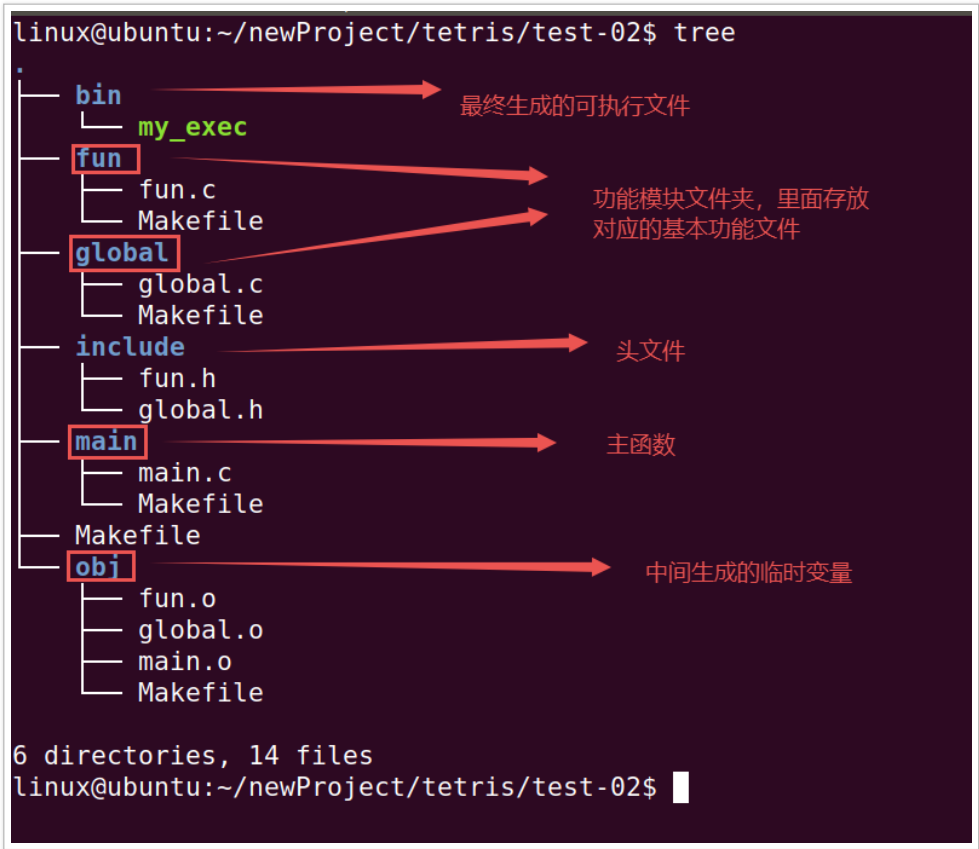


4.4 Makefile 多文件管理乞丐版_物联网 / 嵌入式工程师 - 慕课网

“ 慕课网慕课教程 4.4 Makefile 多文件管理乞丐版涵盖海量编程基础技术教程，以图图文表的形式，把晦涩难懂的编程专业用语，以通俗易懂的方式呈现给用户。

Makefile 主要是对大型多文件工程进行管理，前几个章节，我们跟大家简单的介绍了一下 Makefile 的基本语法。本章节我们来看看针对多工程中 Makfile 的使用方法。一般多工程的目录结构如下：



```
sudo apt-get install tree
```

格式:

```
#在主Makefile中写入内容，调用子Makefile
make -C 子Mafeile的路径
```

示例用法:

```
目录结构:
fun  Makefile
|
|---- fun.c Makefile
```

```
make -C ./fun
```

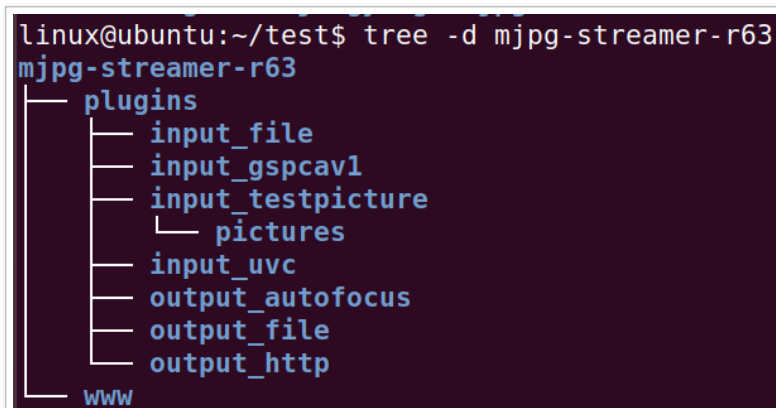
格式:

示例用法:

```
#执行当前的pwd命令
CUR_PATH=${shell pwd}

all :
    @echo $CUR_APTH
```

实际开发中我们不可能只有以上几个文件，我们会把文件分为各种各样的模块来进行管理。以下为一个视频流服务器的工程管理例子。



```
linux@ubuntu:~/test$ tree -d mjpg-streamer-r63
mjpg-streamer-r63
├── plugins
│   ├── input_file
│   ├── input_gspcav1
│   ├── input_testpicture
│   │   └── pictures
│   ├── input_uvc
│   ├── output_autofocus
│   ├── output_file
│   └── output_http
└── www
```

以下每个文件夹中大部分都包含了一个子 Makefile 来对工程代码进行管理。通过主 Makefile 来调用。

```

mjpg-streamer-r63
├── CHANGELOG
├── LICENSE
├── Makefile
├── mjpg_streamer.c
├── mjpg_streamer.h
├── plugins
│   ├── input_file
│   │   └── input_file.c
│   ├── input_gspcav1
│   │   ├── encoder.c
│   │   ├── encoder.h
│   │   ├── huffman.c
│   │   ├── huffman.h
│   │   ├── input_gspcav1.c
│   │   ├── jconfig.h
│   │   ├── jdatatype.h
│   │   └── Makefile
│   ├── input.h
│   ├── input_testpicture
│   │   ├── input_testpicture.c
│   │   └── Makefile
│   └── pictures
│       ├── 160x120_1.jpg
│       ├── 160x120_2.jpg
│       ├── 320x240_1.jpg
│       ├── 320x240_2.jpg
│       ├── 640x480_1.jpg
│       ├── 640x480_2.jpg
│       ├── 960x720_1.jpg
│       └── 960x720_2.jpg
└── testpictures.h
└── input_uvc
    ├── dynctrl.c
    ├── dynctrl.h
    ├── huffman.h
    ├── input_uvc.c
    ├── jpeg_utils.c
    ├── jpeg_utils.h
    ├── Makefile
    └── uvc_compat.h

```

global.h

```

#ifndef __GLOBAL_H__
#define __GLOBAL_H__

#include <stdio.h>

extern int a;

#endif

```

global.c

```

#include "global.h"

int a = 20;

```

fun.h

```

#ifndef __FUN_H__
#define __FUN_H__

#include "fun.h"
#include <stdio.h>
#include "global.h"

extern int fun();

#endif

```

fun.c

```

#include "fun.h"
#include <stdio.h>
#include "global.h"

int fun()
{
    printf("a = %d\n",a);
}

```

main.c

```

#include "fun.h"

int main(int argc, const char *argv[])
{
    fun();
    return 0;
}

```

运行结果

```

gcc *.c -o my_exec
./my_exec

```

- 乞丐版 Makefile 工程架构
 - 首先把自己的代码按照下列架构分别放到对应的文件夹下，并创建空的 Makefile 文件



```

linux@ubuntu:~/newProject/tetris/test$ tree
.
├── fun
│   ├── fun.c
│   └── Makefile
├── global
│   ├── global.c
│   └── Makefile
├── include
│   ├── fun.h
│   └── global.h
├── main
│   ├── main.c
│   └── Makefile
├── Makefile
└── obj
    └── Makefile

```

- 各文件下 Makefile 的编写
 - fun 文件夹下 Makefile 编写
 - `../obj/fun.o:fun.c`
`gcc -c -I ../include/ $< -o $@`
 - global 文件夹下 Makefile 编写
 - `../obj/global.o:global.c`
`gcc -c -I ../include/ $< -o $@`
 - main 文件夹下 Makefile 编写
 - `../obj/main.o:main.c`

```
gcc -c -I ../include/ $< -o $@
```

- obj 文件夹下 Makefile 编写

```
../bin/my_exec:*.o
gcc -I ../include/ $^ -o $@
```

- 主工程架构 Makefile 编写

```
SUB_DIR := main fun global obj
export SUB_DIR

all:$(SUB_DIR)

$(SUB_DIR) : MK_BIN
make -C $@
MK_BIN:
mkdir -p ./bin

clean:
rm -rf ./bin ./obj/*.o
```

- 执行脚本命令

```
make           #生成可执行文件
./bin/my_exec  #执行文件，输出结果
make clean     #清除可执行文件
```

全文完

本文由 简悦 SimpRead 优化，用以提升阅读体验

使用了 全新的简悦词法分析引擎 beta，点击查看详细说明

