# Practice linux kernal module

Danny Deng

Version 1.0

## 1 Command in terminal

### See the command usage and information

```
$ man {command}
```

Example:

```
$ man insmod
```

Result


### Insert a kernal module

```
$ sudo insmod {module name (*.ko)}
```

### Show a kernal module info

```
$ sudo modinfo {module name (*.ko)}
```

### Print or control the kernel ring buffer

```
$ sudo dmesg
```

### Simple program to remove a module from the Linux Kernel

```
$ sudo rmmod {module name}
```

### Add and remove modules from the Linux Kernel

Loads the module only in /lib/modules/‘uname -r‘ modprobe depends on depmod tool to calculate dependancies

```
$ sudo modprobe {module name}
$ echo the dependancies
$ vi /lib/module/‘uname −r‘/modules.dep
```

## 2 Hello world

### Example code below

```
#include <linux/kernel.h>
#include <linux/module.h>

MODULE_LICENSE("GPL");
static int test_hello_init (void) {
    printk(KERN_INFO"%s: In init \n", __func__);
    return 0;
}

static void test_hello_exit (void) {
    printk(KERN_INFO"%s: In exit \n", __func__);
}

module_init( test_hello_init );
module_exit( test_hello_exit );
```

### Include header and define the LICENSE(option)

```
#include <linux/kernel.h>
#include <linux/module.h>

MODULE_LICENSE("GPL");
```

### When insert module, it will call the initial function

```
static int test_hello_init (void) {
    printk(KERN_INFO"%s: In init \n", __func__);
    return 0;
}

module_init( test_hello_init );
```

### When rmmod the kernel module, it will call the exit function

```
static void test_hello_exit (void) {
    printk(KERN_INFO"%s: In exit \n", __func__);
```

```
}

module_exit( test_hello_exit );
```

### Build steps

1. Write the Makefile

```
obj-m := hello_world.o

all:
    make -C /lib/modules/`uname -r`/build M=${PWD} modules

clean:
    make -C /lib/modules/`uname -r`/build M=${PWD} clean
```

2. Build

```
    $ make all
```

# 3 Internel module init

### Example code below

```
#include <linux/kernel.h>
#include <linux/module.h>

MODULE_LICENSE("GPL");

/*
* Kernel module default initial function name
*/
int init_module(void) {
    printk(KERN_INFO"%s: In init\n", __func__);
    return 0;
}

/*
* Kernel module default exit function name
*/
void cleanup_module(void) {
    printk(KERN_INFO"%s: In exit\n", __func__);
}

MODULE_AUTHOR("Danny Deng");
MODULE_DESCRIPTION("Internel module example");
```

- First In my case the *module_init* define in this file "/usr/src/linux-hwe-5.11-headers-5.11.0-41/include/linux/module.h"
- Second, the macro *module_init* will expand the function with "alias" to let function call the same thing with different name.

```
/* Each module must use one module_init(). */
#define module_init(initfn)                \
    static inline initcall_t __maybe_unused __inittest(void)      \
    { return initfn; }                \
    int init_module(void) __copy(initfn) __attribute__((alias(#initfn)));

/* This is only required if you want to be unloadable. */
#define module_exit(exitfn)                \
    static inline exitcall_t __maybe_unused __exittest(void)      \
    { return exitfn; }                \
    void cleanup_module(void) __copy(exitfn) __attribute__((alias(#↩
        exitfn)));

#endif
```

# 4 Passing parameters to linux kernel module

Using *module_param* function to declare parameters
**Example code below**

```
#include <linux/kernel.h>
#include <linux/module.h>

MODULE_LICENSE("GPL");

char *name = "Danny";
int count = 0;
module_param(name, charp, S_IRUGO);
module_param(count, int, S_IRUGO);


static int test_arguments_init(void) {
    printk(KERN_INFO"%s: In init\n", __func__);
    printk(KERN_INFO"%s: name: %s\n", __func__, name);
    printk(KERN_INFO"%s: pass count: %d\n", __func__, count);

    return 0;
}


static void test_exit(void) {
    printk(KERN_INFO"%s: In exit: \n", __func__);
}



module_init(test_arguments_init);
module_exit(test_exit);

MODULE_AUTHOR("Danny Deng");
MODULE_DESCRIPTION("Argument parsing example");
```

**Each parameters can declare different permission**

usr > src > linux-hwe-5.11-headers-5.11.0-41 > include > linux > C stat.h > ...

```c
/* SPDX-License-Identifier: GPL-2.0 */
#ifndef _LINUX_STAT_H
#define _LINUX_STAT_H


#include <asm/stat.h>
#include <uapi/linux/stat.h>

#define S_IRWXUGO   (S_IRWXU|S_IRWXG|S_IRWXO)
#define S_IALLUGO   (S_ISUID|S_ISGID|S_ISVTX|S_IRWXUGO)
#define S_IRUGO     (S_IRUSR|S_IRGRP|S_IROTH)
#define S_IWUGO     (S_IWUSR|S_IWGRP|S_IWOTH)
#define S_IXUGO     (S_IXUSR|S_IXGRP|S_IXOTH)
```

**The macro value define below**

```c
#define S_IRWXU 00700
#define S_IRUSR 00400
#define S_IWUSR 00200
#define S_IXUSR 00100

#define S_IRWXG 00070
#define S_IRGRP 00040
#define S_IWGRP 00020
#define S_IXGRP 00010

#define S_IRWXO 00007
#define S_IROTH 00004
#define S_IWOTH 00002
#define S_IXOTH 00001
```

S_I + R/W/X + USR/GRP/OTH

- R: Read
- W: Write
- X: Execute
- USR: User
- GRP: Group
- OTH: Other

**Insert the kernel module, and feed the parameter**

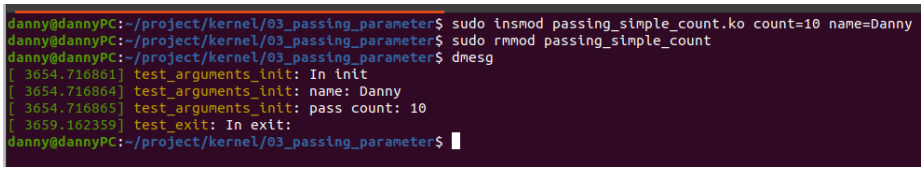
```
$ sudo insmod passing_simple_count.ko count=10 name=Danny
```

**Remove the kernel module**

```
$ sudo rmsmod passing_simple_count
```

**Show the result**

```
$ dmesg
```




# 5 Module stacking

**Module 1 code below, declare the my_add function**
Using *EXPORT_SYMBOL* to export symbol

```
#include <linux/kernel.h>
#include <linux/module.h>

MODULE_LICENSE("GPL");

int my_add(int a, int b) {
    return a + b;
}
EXPORT_SYMBOL(my_add);

static int test_init (void) {
    printk(KERN_INFO"%s: In init by module1\n", __func__);

    return 0;
}

static void test_exit (void) {
    printk(KERN_INFO"%s: In exit by module1: \n", __func__);
}

module_init( test_init );
module_exit( test_exit );

MODULE_AUTHOR("Danny Deng");
MODULE_DESCRIPTION("Argument parsing example");
```

**Module 2 code below, call the my_add function**

```
#include <linux/kernel.h>
#include <linux/module.h>

MODULE_LICENSE("GPL");

/* Define in module1.c and already EXPORT_SYMBOL */
extern int my_add(int a, int b);

static int test_init (void) {
    printk(KERN_INFO"%s: In init by module2\n", __func__);
    printk(KERN_INFO"my_add(1, 2): %d\n", my_add(1, 2));
```

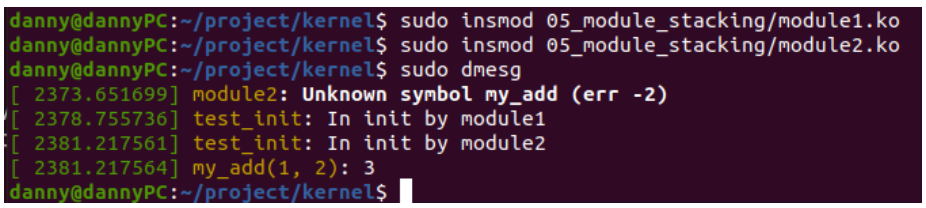
```
    return 0;
}

static void test_exit (void) {
    printk(KERN_INFO"%s: In exit by module2: \n", __func__);
}

module_init( test_init );
module_exit( test_exit );

MODULE_AUTHOR("Danny Deng");
MODULE_DESCRIPTION("Argument parsing example");
```

**Result**




# 6 Module licence

## 6.1 Tainted kernel document

Linux bash to determine whether tainted kernel

- tainted kernel check script
- tainted kernels document

**Use the run time command to determine whether kernel tainted**

```
$ cat /proc/sys/kernel/tainted
```

## 6.2 Result