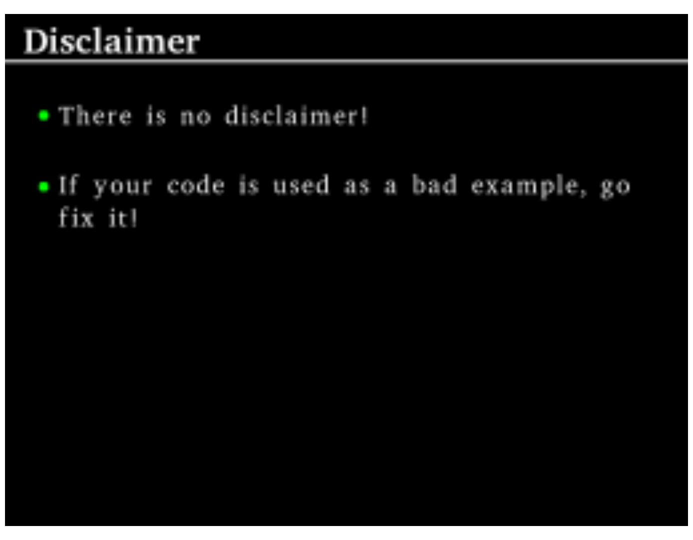
# Documentation/CodingStyle
and beyond...

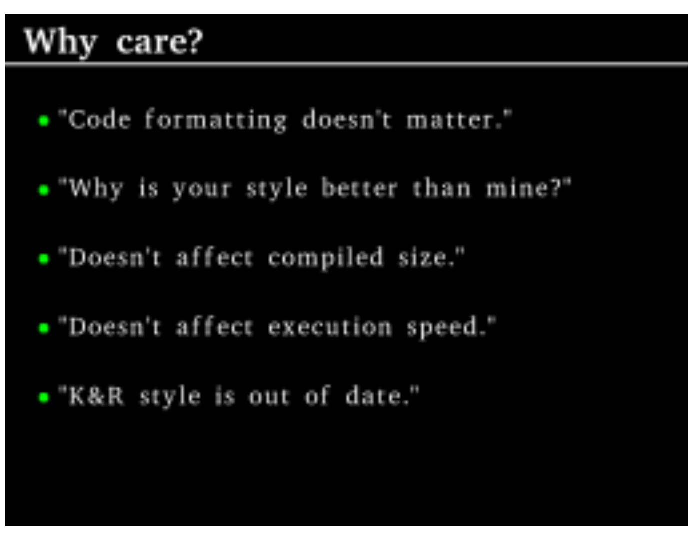
Greg Kroah-Hartman

greg@kroah.com
gregkh@us.ibm.com



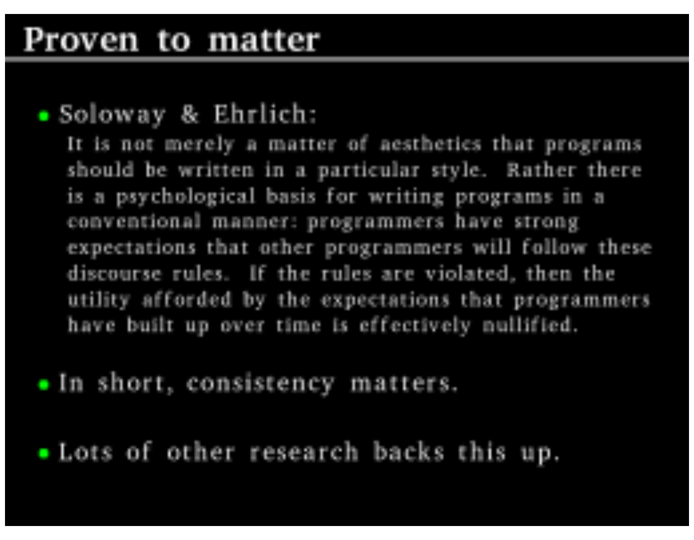
## Disclaimer

- There is no disclaimer!
- If your code is used as a bad example, go fix it!



## Why care?

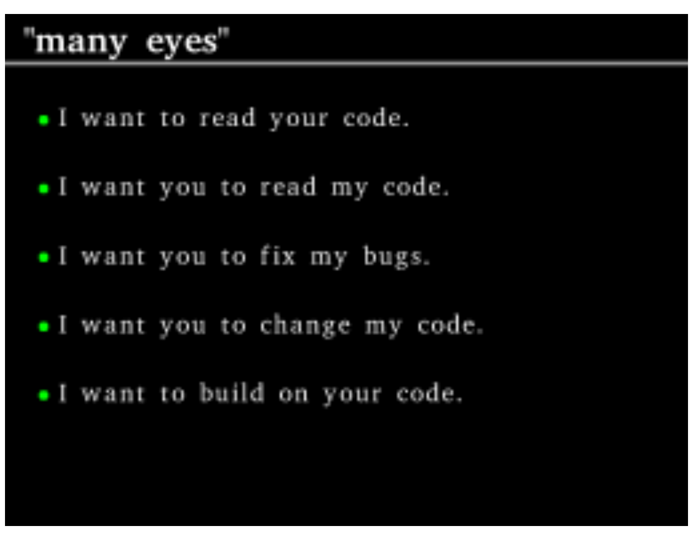
- "Code formatting doesn't matter."
- "Why is your style better than mine?"
- "Doesn't affect compiled size."
- "Doesn't affect execution speed."
- "K&R style is out of date."



## Proven to matter

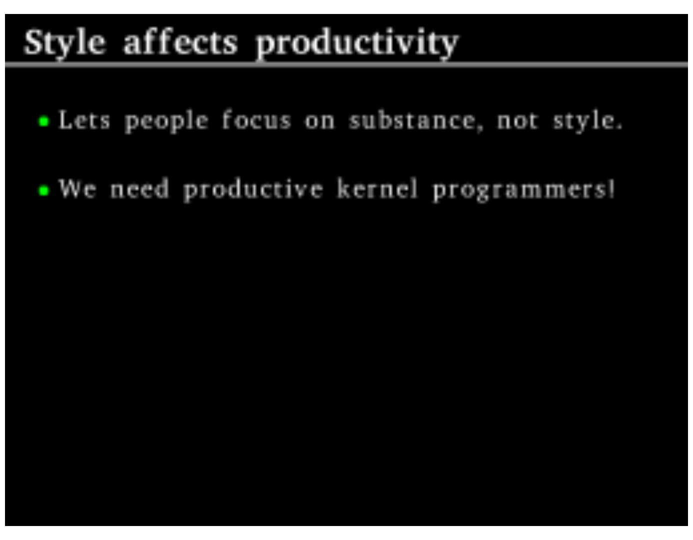
- Soloway & Ehrlich:

  It is not merely a matter of aesthetics that programs should be written in a particular style. Rather there is a psychological basis for writing programs in a conventional manner: programmers have strong expectations that other programmers will follow these discourse rules. If the rules are violated, then the utility afforded by the expectations that programmers have built up over time is effectively nullified.
- In short, consistency matters.
- Lots of other research backs this up.



## "many eyes"

- I want to read your code.
- I want you to read my code.
- I want you to fix my bugs.
- I want you to change my code.
- I want to build on your code.

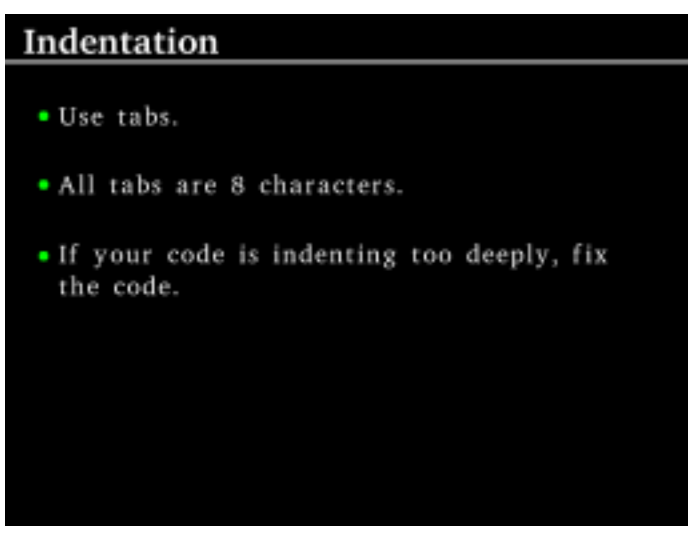


## Style affects productivity

- Lets people focus on substance, not style.
- We need productive kernel programmers!

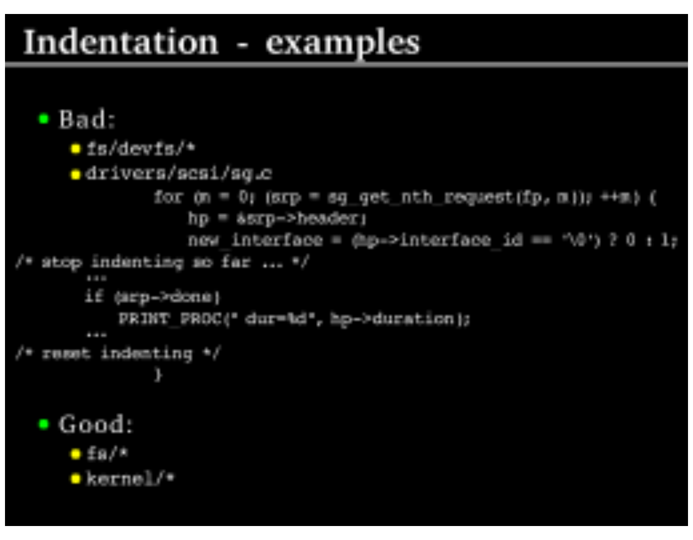
## So what are the rules?

Documentation/CodingStyle



## Indentation

- Use tabs.
- All tabs are 8 characters.
- If your code is indenting too deeply, fix the code.

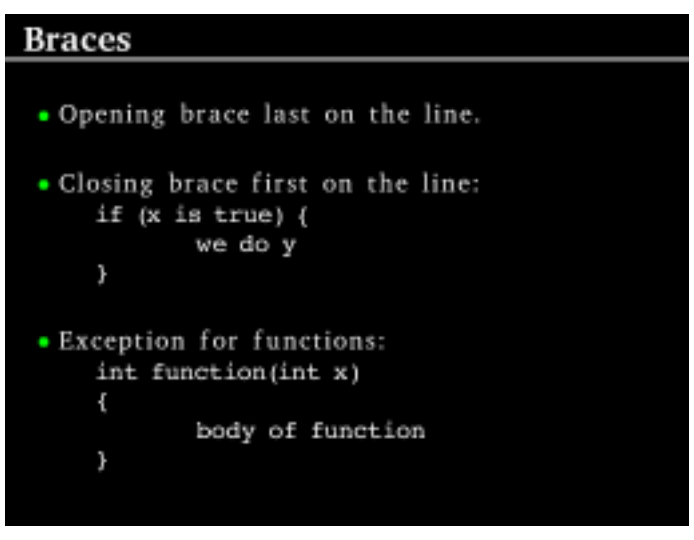


## Indentation - examples

- Bad:
  - fs/devfs/*
  - drivers/scsi/sg.c

```
        for (m = 0; (srp = sg_get_nth_request(fp, m)); ++m) {
            hp = &srp->header;
            new_interface = (hp->interface_id == '\0') ? 0 : 1;
/* stop indenting so far ... */
    ...
    if (srp->done)
        PRINT_PROC(" dur=%d", hp->duration);
    ...
/* reset indenting */
        }
```

- Good:
  - fs/*
  - kernel/*



## Braces

- Opening brace last on the line.
- Closing brace first on the line:

```
if (x is true) {
        we do y
}
```

- Exception for functions:

```
int function(int x)
{
        body of function
}
```

## Braces - examples

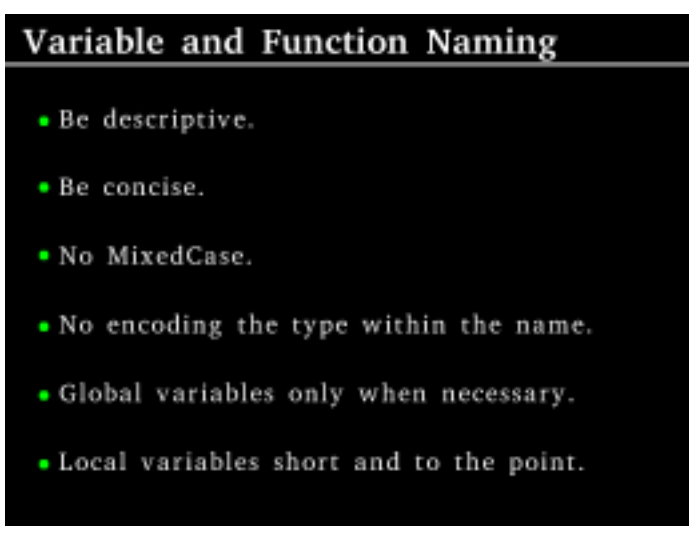
- Bad:
  - fs/devfs/*

```
if (type == DEVFS_SPECIAL_CHR)
{
    semaphore = &char_semaphore;
    list = &char_list;
}
else
{
    semaphore = &block_semaphore;
    list = &block_list;
}
```

- Good:
  - kernel/*.c
  - drivers/scsi/qla1280.c
    - thanks to Jes Sorensen

## Automatic code fixer

scripts/Lindent

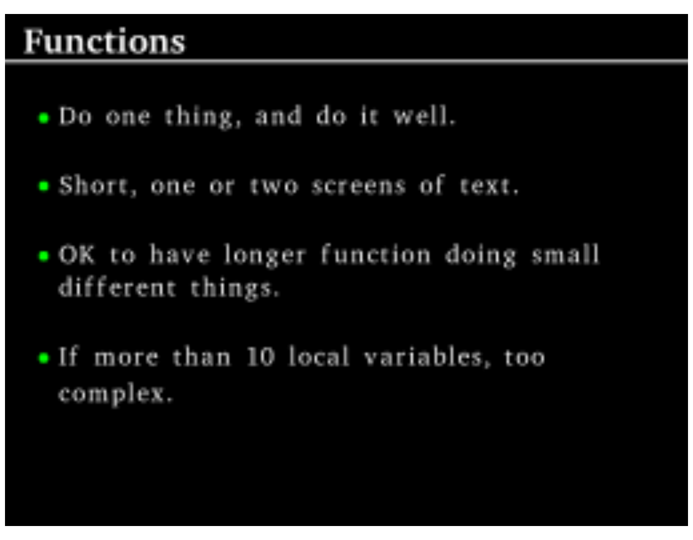


## Variable and Function Naming

- Be descriptive.
- Be concise.
- No MixedCase.
- No encoding the type within the name.
- Global variables only when necessary.
- Local variables short and to the point.

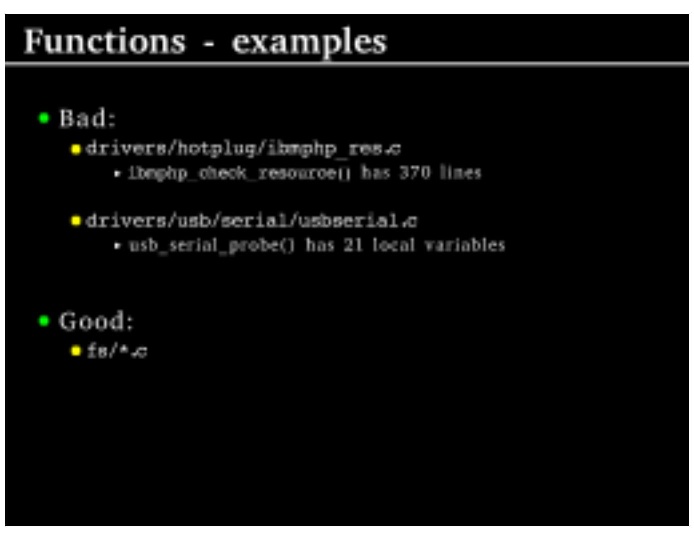
## Naming - examples

- Bad:
  - CommandAllocationGroupSize
  - DAC960_V1_EnableMemoryMailboxInterface()
  - loop_counter
  - drivers/bock/DAC960.*
- Good:
  - cmd_group_size
  - enable_mem_mailbox()
  - i
  - fs/*.c



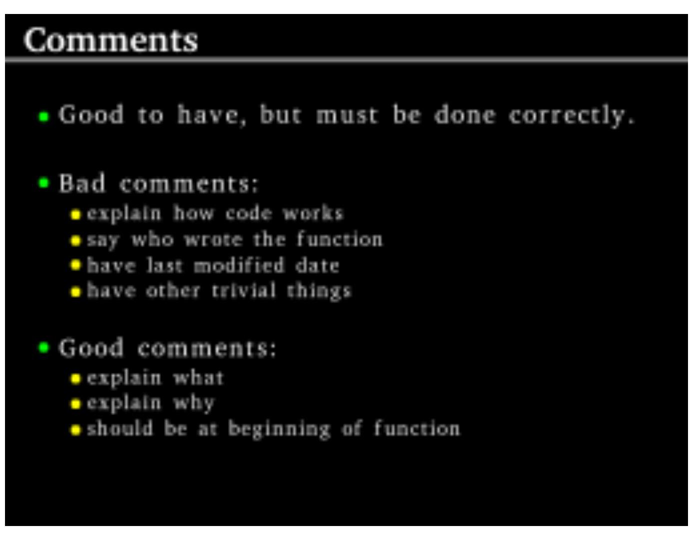
## Functions

- Do one thing, and do it well.
- Short, one or two screens of text.
- OK to have longer function doing small different things.
- If more than 10 local variables, too complex.



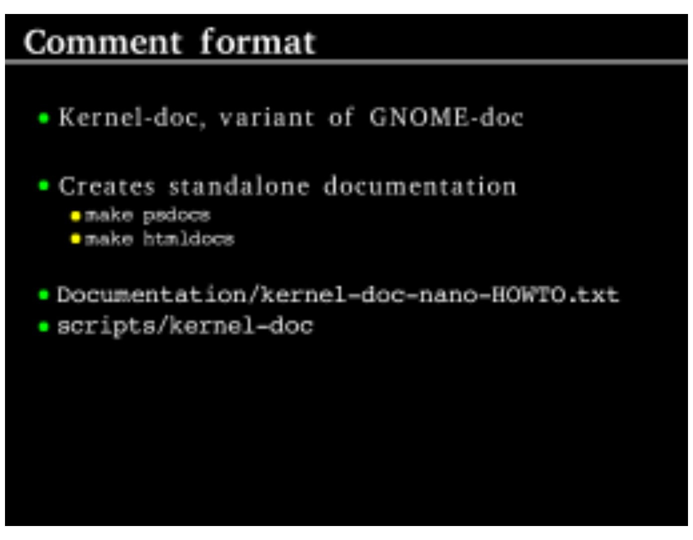
## Functions - examples

- Bad:
  - drivers/hotplug/ibmphp_res.c
    - ibmphp_check_resource() has 370 lines
  - drivers/usb/serial/usbserial.c
    - usb_serial_probe() has 21 local variables
- Good:
  - fs/*.c



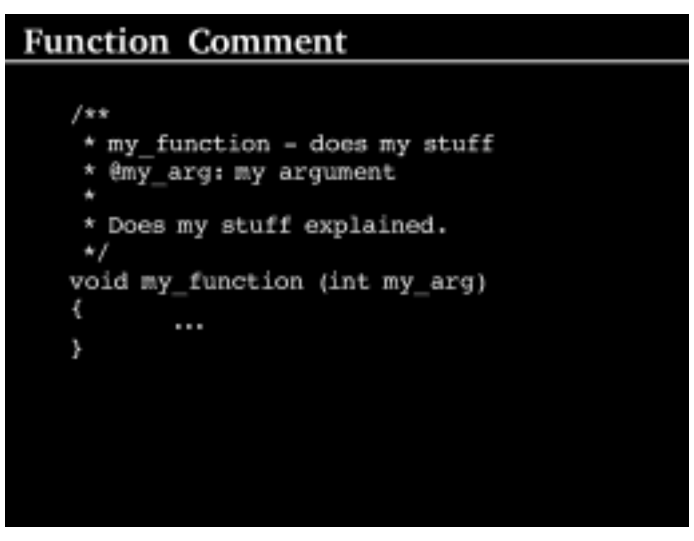
## Comments

- Good to have, but must be done correctly.
- Bad comments:
  - explain how code works
  - say who wrote the function
  - have last modified date
  - have other trivial things
- Good comments:
  - explain what
  - explain why
  - should be at beginning of function



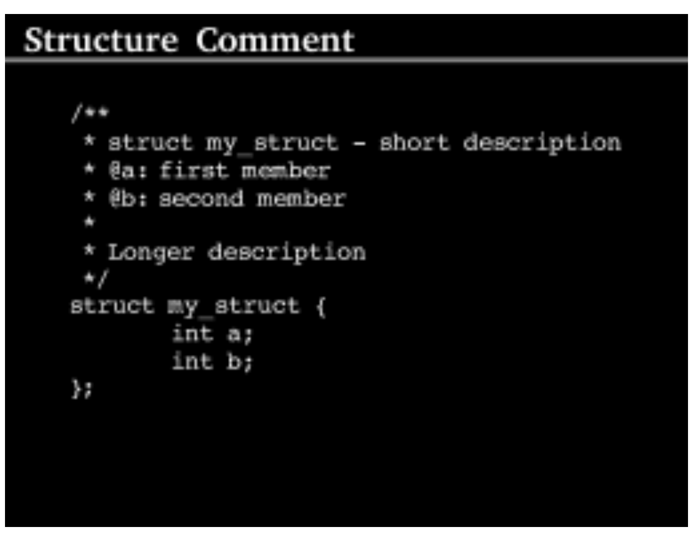
## Comment format

- Kernel-doc, variant of GNOME-doc
- Creates standalone documentation
  - make psdocs
  - make htmldocs
- Documentation/kernel-doc-nano-HOWTO.txt
- scripts/kernel-doc



## Function Comment

```
/**
 * my_function - does my stuff
 * @my_arg: my argument
 *
 * Does my stuff explained.
 */
void my_function (int my_arg)
{
        ...
}
```

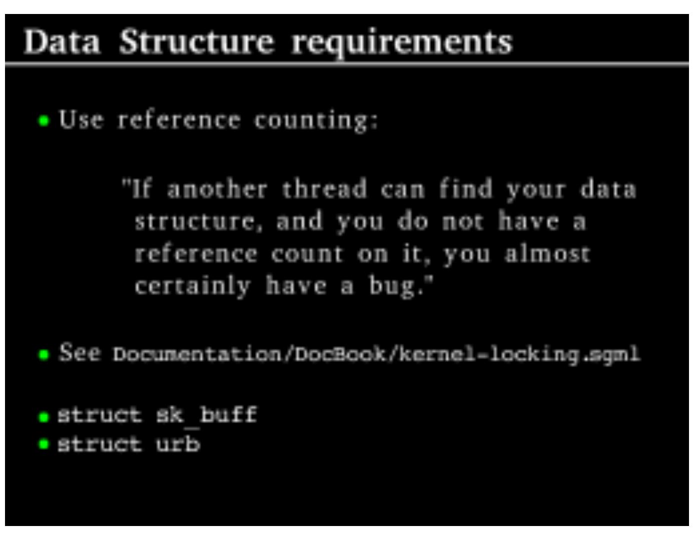


## Structure Comment

```
/**
 * struct my_struct - short description
 * @a: first member
 * @b: second member
 *
 * Longer description
 */
struct my_struct {
        int a;
        int b;
};
```

## Comments - examples

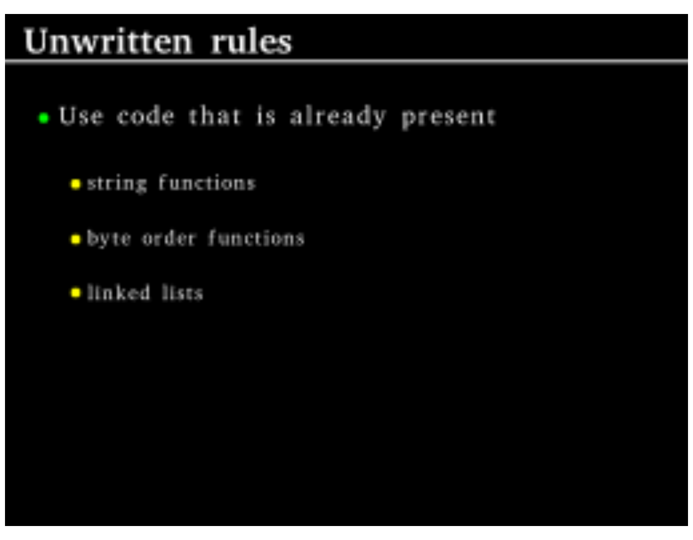
- Bad:
  - arch/i386/kernel/mtrr.c
    - mix of old and new style comments
  - drivers/scsi/pci2220i.c
    - how to NOT create function comment blocks
- Good:
  - drivers/usb/core/*.c



## Data Structure requirements

- Use reference counting:

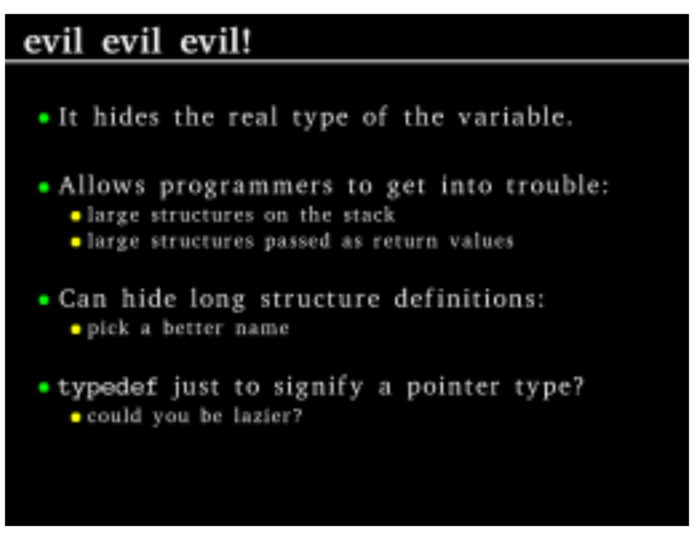
  > "If another thread can find your data structure, and you do not have a reference count on it, you almost certainly have a bug."
- See Documentation/DocBook/kernel-locking.sgml
- struct sk_buff
- struct urb



## Unwritten rules

- Use code that is already present
  - string functions
  - byte order functions
  - linked lists

typedef is evil

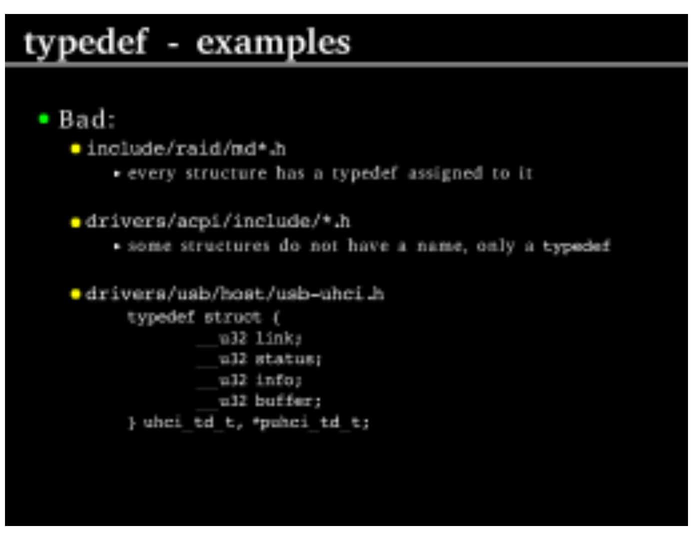


## evil evil evil!

- It hides the real type of the variable.
- Allows programmers to get into trouble:
  - large structures on the stack
  - large structures passed as return values
- Can hide long structure definitions:
  - pick a better name
- typedef just to signify a pointer type?
  - could you be lazier?

## Well, mostly evil

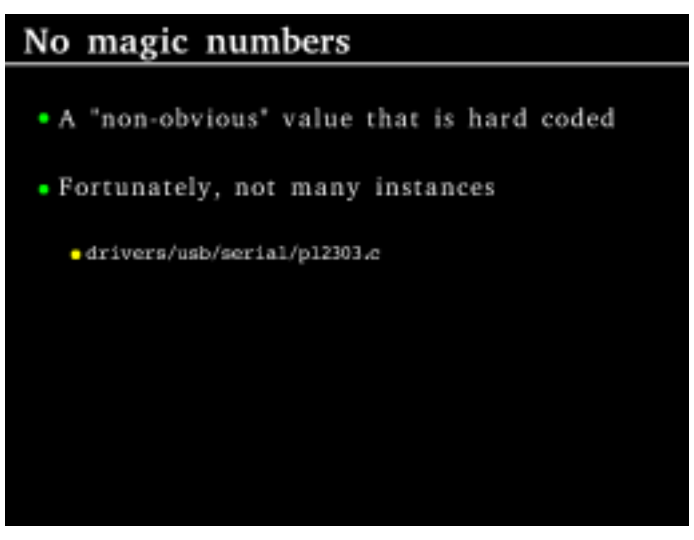
- Base system types
  - u8, u16, u32, u64, etc.
  - dev_t
  - list_t
- Function pointers



## typedef - examples

- Bad:
  - include/raid/md*.h
    - every structure has a typedef assigned to it
  - drivers/acpi/include/*.h
    - some structures do not have a name, only a typedef
  - drivers/usb/host/usb-uhci.h

```
typedef struct {
        __u32 link;
        __u32 status;
        __u32 info;
        __u32 buffer;
} uhci_td_t, *puhci_td_t;
```



## No magic numbers

- A "non-obvious" value that is hard coded
- Fortunately, not many instances
  - drivers/usb/serial/pl2303.c

## No #ifdef in .c files

- #ifdef belongs in .h file
- Let the compiler do the work

## No #ifdef in .c files - example

- Before:
  - drivers/usb/hid_core.c

```
static void hid_process_event (struct hid_device *hid, struct hid_field *field
,
                               struct hid_usage *usage, __s32 value)
{
        hid_dump_input(usage, value);
        if (hid->claimed & HID_CLAIMED_INPUT)
                hidinput_hid_event(hid, field, usage, value);
#ifdef CONFIG_USB_HIDDEV
        if (hid->claimed & HID_CLAIMED_HIDDEV)
                hiddev_hid_event(hid, usage->hid, value);
#endif
}
```

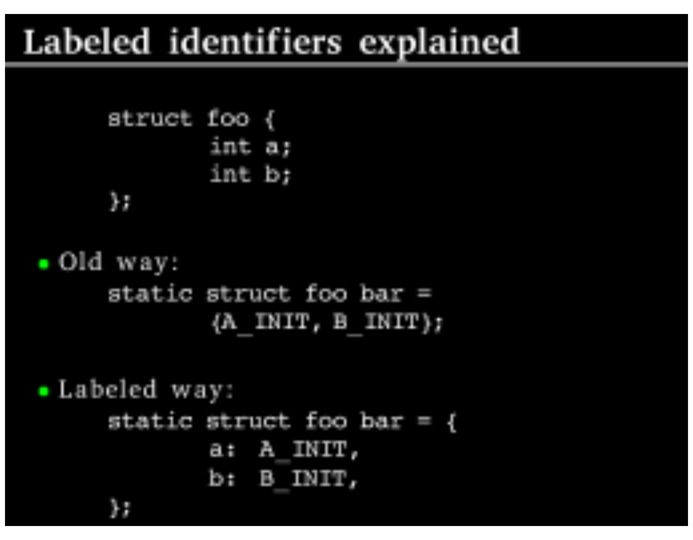
## No #ifdef in .c files - example

- After:
  - include/linux/hiddev.h

```
#ifdef CONFIG_USB_HIDDEV
        extern void hiddev_hid_event (struct hid_device *, unsigned int usage,
                                      int value);
#else
        static inline void hiddev_hid_event (struct hid_device *hid,
                                             unsigned int usage, int value)
                                             { }
#endif
```

  - drivers/usb/hid_core.c

```
static void hid_process_event (struct hid_device *hid, struct hid_field *field
,
                               struct hid_usage *usage, __s32 value)
{
        hid_dump_input(usage, value);
        if (hid->claimed & HID_CLAIMED_INPUT)
                hidinput_hid_event(hid, field, usage, value);
        if (hid->claimed & HID_CLAIMED_HIDDEV)
                hiddev_hid_event(hid, usage->hid, value);
}
```



## Labeled identifiers explained

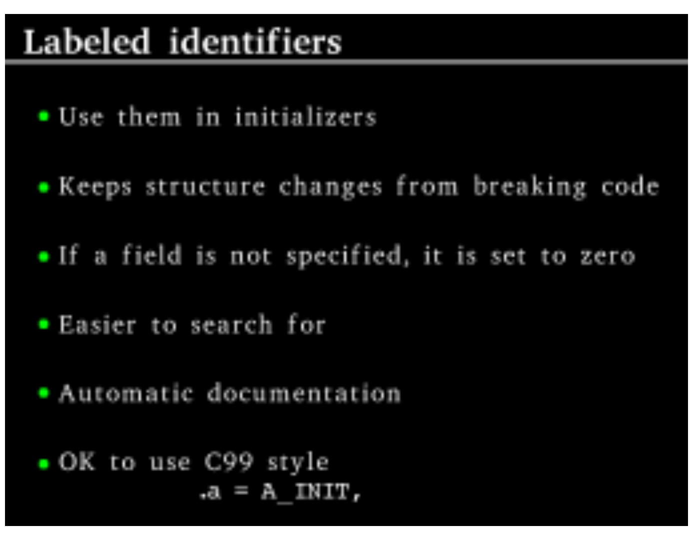
```
struct foo {
        int a;
        int b;
};
```

- Old way:

```
static struct foo bar =
        {A_INIT, B_INIT};
```

- Labeled way:

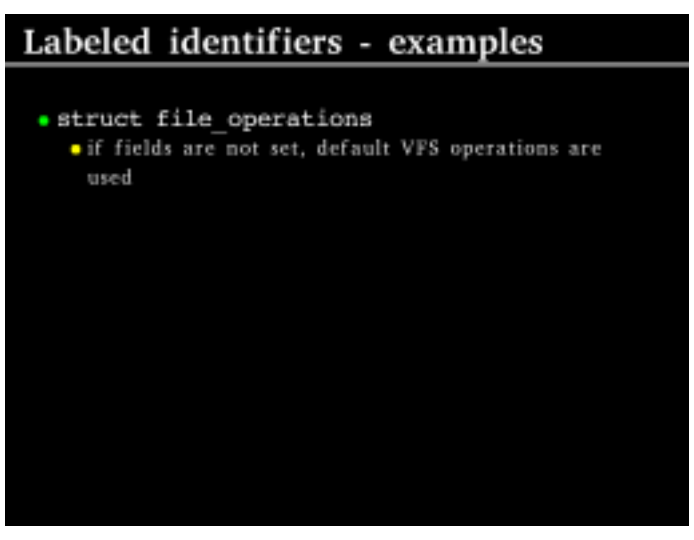
```
static struct foo bar = {
        a:  A_INIT,
        b:  B_INIT,
};
```



## Labeled identifiers

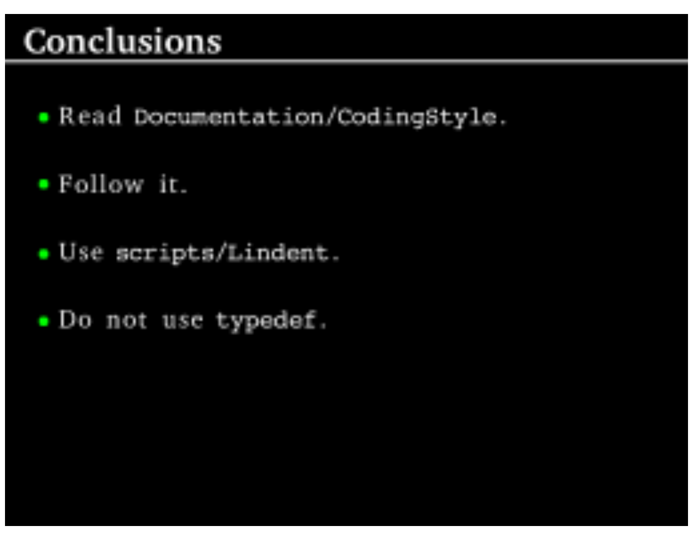
- Use them in initializers
- Keeps structure changes from breaking code
- If a field is not specified, it is set to zero
- Easier to search for
- Automatic documentation
- OK to use C99 style

```
        .a = A_INIT,
```



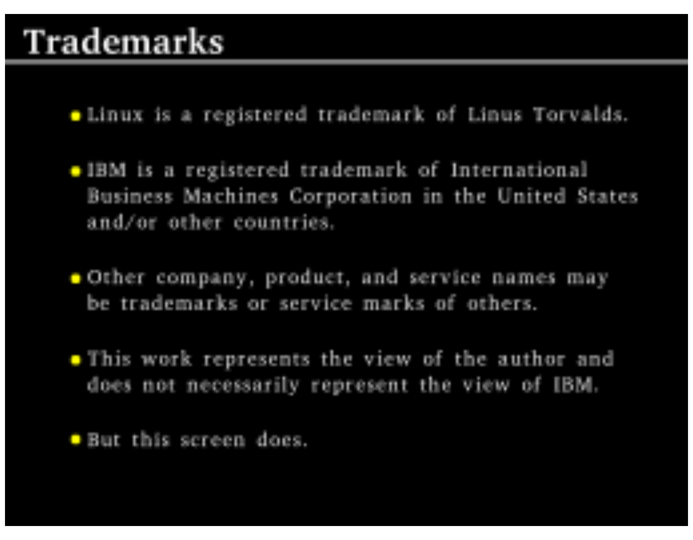
## Labeled identifiers - examples

- struct file_operations
  - if fields are not set, default VFS operations are used



## Conclusions

- Read Documentation/CodingStyle.
- Follow it.
- Use scripts/Lindent.
- Do not use typedef.



## Trademarks

- Linux is a registered trademark of Linus Torvalds.
- IBM is a registered trademark of International Business Machines Corporation in the United States and/or other countries.
- Other company, product, and service names may be trademarks or service marks of others.
- This work represents the view of the author and does not necessarily represent the view of IBM.
- But this screen does.

# Documentation/CodingStyle
and beyond...

Greg Kroah-Hartman

greg@kroah.com
gregkh@us.ibm.com

Generated by MagicPoint