Checkpatch

Checkpatch (scripts/checkpatch.pl) is a perl script which checks for trivial style violations in patches and optionally corrects them. Checkpatch can also be run on file contexts and without the kernel tree.

Checkpatch is not always right. Your judgement takes precedence over checkpatch messages. If your code looks better with the violations, then its probably best left alone.

Options

This section will describe the options checkpatch can be run with.

Usage:

```
./scripts/checkpatch.pl [OPTION]... [FILE]...
```

Available options:

- -q, --quiet

  Enable quiet mode.

- -v, --verbose Enable verbose mode. Additional verbose test descriptions are output so as to provide information on why that particular message is shown.

- --no-tree

  Run checkpatch without the kernel tree.

- --no-signoff

  Disable the 'Signed-off-by' line check. The sign-off is a simple line at the end of the explanation for the patch, which certifies that you wrote it or otherwise have the right to pass it on as an open-source patch.

  Example:

  ```
  Signed-off-by: Random J Developer <random@developer.example.org>
  ```

  Setting this flag effectively stops a message for a missing signed-off-by line in a patch context.

- --patch

  Treat FILE as a patch. This is the default option and need not be explicitly specified.

- --emacs

  Set output to emacs compile window format. This allows emacs users to jump from the error in the compile window directly to the offending line in the patch.

- --terse

  Output only one line per report.

- --showfile

  Show the diffed file position instead of the input file position.

- -g, --git

  Treat FILE as a single commit or a git revision range.

  Single commit with:

  - <rev>
  - <rev>^
  - <rev>~n

  Multiple commits with:

  - <rev1>..<rev2>
  - <rev1>...<rev2>
  - <rev>-<count>

- -f, --file

  Treat FILE as a regular source file. This option must be used when running checkpatch on source files in the kernel.

- --subjective, --strict

  Enable stricter tests in checkpatch. By default the tests emitted as CHECK do not activate by default. Use this flag to activate the CHECK tests.

- --list-types

  Every message emitted by checkpatch has an associated TYPE. Add this flag to display all the types in checkpatch.

  Note that when this flag is active, checkpatch does not read the input FILE, and no message is emitted. Only a list of types in checkpatch is output.

- --types TYPE(,TYPE2...)

  Only display messages with the given types.

  Example:

  ```
  ./scripts/checkpatch.pl mypatch.patch --types EMAIL_SUBJECT,BRACES
  ```

- --ignore TYPE(,TYPE2...)

  Checkpatch will not emit messages for the specified types.

  Example:

  ```
  ./scripts/checkpatch.pl mypatch.patch --ignore EMAIL_SUBJECT,BRACES
  ```

- --show-types

  By default checkpatch doesn't display the type associated with the messages. Set this flag to show the message type in the output.

- --max-line-length=n

  Set the max line length (default 100). If a line exceeds the specified length, a LONG_LINE message is emitted.

  The message level is different for patch and file contexts. For patches, a WARNING is emitted. While a milder CHECK is emitted for files. So for file contexts, the --strict flag must also be enabled.

- --min-conf-desc-length=n

  Set the Kconfig entry minimum description length, if shorter, warn.

- --tab-size=n

  Set the number of spaces for tab (default 8).

- --root=PATH

  PATH to the kernel tree root.

  This option must be specified when invoking checkpatch from outside the kernel root.

- --no-summary

  Suppress the per file summary.

- --mailback

  Only produce a report in case of Warnings or Errors. Milder Checks are excluded from this.

- --summary-file

  Include the filename in summary.

- --debug KEY=[0|1]

  Turn on/off debugging of KEY, where KEY is one of 'values', 'possible', 'type', and 'attr' (default is all off).

- --fix

  This is an EXPERIMENTAL feature. If correctable errors exists, a file <inputfile>.EXPERIMENTAL-checkpatch-fixes is created which has the automatically fixable errors corrected.

- --fix-inplace

  EXPERIMENTAL - Similar to --fix but input file is overwritten with fixes.

  DO NOT USE this flag unless you are absolutely sure and you have a backup in place.

- --ignore-perl-version

  Override checking of perl version. Runtime errors maybe encountered after enabling this flag if the perl version does not meet the minimum specified.

- --codespell

  Use the codespell dictionary for checking spelling errors.

- --codespellfile

  Use the specified codespell file. Default is '/usr/share/codespell/dictionary.txt'.

- --typedefsfile

Read additional types from this file.

- --color[=WHEN]

  Use colors 'always', 'never', or only when output is a terminal ('auto'). Default is 'auto'.

- --kconfig-prefix=WORD

  Use WORD as a prefix for Kconfig symbols (default is *CONFIG_*).

- -h, --help, --version

  Display the help text.

Message Levels

Messages in checkpatch are divided into three levels. The levels of messages in checkpatch denote the severity of the error. They are:

- ERROR

  This is the most strict level. Messages of type ERROR must be taken seriously as they denote things that are very likely to be wrong.

- WARNING

  This is the next stricter level. Messages of type WARNING requires a more careful review. But it is milder than an ERROR.

- CHECK

  This is the mildest level. These are things which may require some thought.

Type Descriptions

This section contains a description of all the message types in checkpatch.

Allocation style

ALLOC_ARRAY_ARGS

The first argument for kcalloc or kmalloc_array should be the number of elements. sizeof() as the first argument is generally wrong.

See: https://www.kernel.org/doc/html/latest/core-api/memory-allocation.html

ALLOC_SIZEOF_STRUCT

The allocation style is bad. In general for family of allocation functions using sizeof() to get memory size, constructs like:

```
p = alloc(sizeof(struct foo), ...)
```

should be:

```
p = alloc(sizeof(*p), ...)
```

See: https://www.kernel.org/doc/html/latest/process/coding-style.html#allocating-memory

ALLOC_WITH_MULTIPLY

Prefer kmalloc_array/kcalloc over kmalloc/kzalloc with a sizeof multiply.

See: https://www.kernel.org/doc/html/latest/core-api/memory-allocation.html

API usage

ARCH_DEFINES

Architecture specific defines should be avoided wherever possible.

ARCH_INCLUDE_LINUX

Whenever asm/file.h is included and linux/file.h exists, a conversion can be made when linux/file.h includes asm/file.h. However this is not always the case (See signal.h). This message type is emitted only for includes from arch/.

AVOID_BUG

BUG() or BUG_ON() should be avoided totally. Use WARN() and WARN_ON() instead, and handle the "impossible" error condition as gracefully as possible.

See: https://www.kernel.org/doc/html/latest/process/deprecated.html#bug-and-bug-on

CONSIDER_KSTRTO

The simple_strtol(), simple_strtoll(), simple_strtoul(), and simple_strtoull() functions explicitly ignore overflows, which may lead to unexpected results in callers. The respective kstrtol(), kstrtoll(), kstrtoul(), and kstrtoull() functions tend to be the correct replacements.

See: https://www.kernel.org/doc/html/latest/process/deprecated.html#simple-strtol-simple-strtoll-simple-strtoul-simple-strtoull

CONSTANT_CONVERSION

Use of __constant_<foo> form is discouraged for the following functions:

```
__constant_cpu_to_be[x]
__constant_cpu_to_le[x]
__constant_be[x]_to_cpu
__constant_le[x]_to_cpu
__constant_htons
__constant_ntohs
```

Using any of these outside of include/uapi/ is not preferred as using the function without __constant_ is identical when the argument is a constant.

In big endian systems, the macros like __constant_cpu_to_be32(x) and cpu_to_be32(x) expand to the same expression:

```
#define __constant_cpu_to_be32(x) ((__force __be32)(__u32)(x))
#define __cpu_to_be32(x)          ((__force __be32)(__u32)(x))
```

In little endian systems, the macros __constant_cpu_to_be32(x) and cpu_to_be32(x) expand to __constant_swab32 and __swab32. __swab32 has a __builtin_constant_p check:

```
#define __swab32(x)                                     \
  (__builtin_constant_p((__u32)(x)) ?     \
  ___constant_swab32(x) :                 \
  __fswab32(x))
```

So ultimately they have a special case for constants. Similar is the case with all of the macros in the list. Thus using the __constant_... forms are unnecessarily verbose and not preferred outside of include/uapi.

See: https://lore.kernel.org/lkml/1400106425.12666.6.camel@joe-AO725/

DEPRECATED_API

Usage of a deprecated RCU API is detected. It is recommended to replace old flavourful RCU APIs by their new vanilla-RCU counterparts.

The full list of available RCU APIs can be viewed from the kernel docs.

See: https://www.kernel.org/doc/html/latest/RCU/whatisRCU.html#full-list-of-rcu-apis

DEPRECATED_VARIABLE

EXTRA_{A,C,CPP,LD}FLAGS are deprecated and should be replaced by the new flags added via commit f77bf01425b1 ("kbuild: introduce ccflags-y, asflags-y and ldflags-y").

The following conversion scheme maybe used:

```
EXTRA_AFLAGS    ->  asflags-y
EXTRA_CFLAGS    ->  ccflags-y
EXTRA_CPPFLAGS  ->  cppflags-y
EXTRA_LDFLAGS   ->  ldflags-y
```

See:

1. https://lore.kernel.org/lkml/20070930191054.GA15876@uranus.ravnborg.org/
2. https://lore.kernel.org/lkml/1313384834-24433-12-git-send-email-lacombar@gmail.com/
3. https://www.kernel.org/doc/html/latest/kbuild/makefiles.html#compilation-flags

DEVICE_ATTR_FUNCTIONS

The function names used in DEVICE_ATTR is unusual. Typically, the store and show functions are used with <attr>_store and <attr>_show, where <attr> is a named attribute variable of the device.

Consider the following examples:

```
static DEVICE_ATTR(type, 0444, type_show, NULL);
static DEVICE_ATTR(power, 0644, power_show, power_store);
```

The function names should preferably follow the above pattern.

See: https://www.kernel.org/doc/html/latest/driver-api/driver-model/device.html#attributes

DEVICE_ATTR_RO

The DEVICE_ATTR_RO(name) helper macro can be used instead of DEVICE_ATTR(name, 0444, name_show, NULL);

Note that the macro automatically appends _show to the named attribute variable of the device for the show method.

See: https://www.kernel.org/doc/html/latest/driver-api/driver-model/device.html#attributes

DEVICE_ATTR_RW

The DEVICE_ATTR_RW(name) helper macro can be used instead of DEVICE_ATTR(name, 0644, name_show, name_store);

Note that the macro automatically appends _show and _store to the named attribute variable of the device for the show and store methods.

See: https://www.kernel.org/doc/html/latest/driver-api/driver-model/device.html#attributes

DEVICE_ATTR_WO

The DEVICE_AATR_WO(name) helper macro can be used instead of DEVICE_ATTR(name, 0200, NULL, name_store);

Note that the macro automatically appends _store to the named attribute variable of the device for the store method.

See: https://www.kernel.org/doc/html/latest/driver-api/driver-model/device.html#attributes

DUPLICATED_SYSCTL_CONST

Commit d91bff3011cf ("proc/sysctl: add shared variables for range check") added some shared const variables to be used instead of a local copy in each source file.

Consider replacing the sysctl range checking value with the shared one in include/linux/sysctl.h. The following conversion scheme may be used:

```
&zero     ->  SYSCTL_ZERO
&one      ->  SYSCTL_ONE
&int_max  ->  SYSCTL_INT_MAX
```

See:

1. https://lore.kernel.org/lkml/20190430180111.10688-1-mcroce@redhat.com/
2. https://lore.kernel.org/lkml/20190531131422.14970-1-mcroce@redhat.com/

ENOSYS

ENOSYS means that a nonexistent system call was called. Earlier, it was wrongly used for things like invalid operations on otherwise valid syscalls. This should be avoided in new code.

See:
https://lore.kernel.org/lkml/5eb299021dec23c1a48fa7d9f2c8b794e967766d.1408730669.git.luto@amacapital.net/

ENOTSUPP

ENOTSUPP is not a standard error code and should be avoided in new patches. EOPNOTSUPP should be used instead.

See: https://lore.kernel.org/netdev/20200510182252.GA411829@lunn.ch/

EXPORT_SYMBOL

EXPORT_SYMBOL should immediately follow the symbol to be exported.

IN_ATOMIC

in_atomic() is not for driver use so any such use is reported as an ERROR. Also in_atomic() is often used to determine if sleeping is permitted, but it is not reliable in this use model. Therefore its use is strongly discouraged.

However, in_atomic() is ok for core kernel use.

See: https://lore.kernel.org/lkml/20080320201723.b87b3732.akpm@linux-foundation.org/

LOCKDEP

The lockdep_no_validate class was added as a temporary measure to prevent warnings on conversion of device->sem to device->mutex. It should not be used for any other purpose.

See: https://lore.kernel.org/lkml/1268959062.9440.467.camel@laptop/

MALFORMED_INCLUDE

The #include statement has a malformed path. This has happened because the author has included a double slash "//" in the pathname accidentally.

USE_LOCKDEP

lockdep_assert_held() annotations should be preferred over assertions based on spin_is_locked()

See: https://www.kernel.org/doc/html/latest/locking/lockdep-design.html#annotations

UAPI_INCLUDE

No #include statements in include/uapi should use a uapi/ path.

USLEEP_RANGE

usleep_range() should be preferred over udelay(). The proper way of using usleep_range() is mentioned in the kernel docs.

See: https://www.kernel.org/doc/html/latest/timers/timers-howto.html#delays-information-on-the-various-kernel-delay-sleep-mechanisms

Comments

BLOCK_COMMENT_STYLE

The comment style is incorrect. The preferred style for multi- line comments is:

```
/*
* This is the preferred style
* for multi line comments.
*/
```

The networking comment style is a bit different, with the first line not empty like the former:

```
/* This is the preferred comment style
* for files in net/ and drivers/net/
*/
```

See: https://www.kernel.org/doc/html/latest/process/coding-style.html#commenting

C99_COMMENTS

C99 style single line comments (//) should not be used. Prefer the block comment style instead.

See: https://www.kernel.org/doc/html/latest/process/coding-style.html#commenting

DATA_RACE

Applications of data_race() should have a comment so as to document the reasoning behind why it was deemed safe.

See: https://lore.kernel.org/lkml/20200401101714.44781-1-elver@google.com/

FSF_MAILING_ADDRESS

Kernel maintainers reject new instances of the GPL boilerplate paragraph directing people to write to the FSF for a copy of the GPL, since the FSF has moved in the past and may do so again. So do not write paragraphs about writing to the Free Software Foundation's mailing address.

See: https://lore.kernel.org/lkml/20131006222342.GT19510@leaf/

Commit message

BAD_SIGN_OFF

The signed-off-by line does not fall in line with the standards specified by the community.

See: https://www.kernel.org/doc/html/latest/process/submitting-patches.html#developer-s-certificate-of-origin-1-1

BAD_STABLE_ADDRESS_STYLE

The email format for stable is incorrect. Some valid options for stable address are:

```
1. stable@vger.kernel.org
2. stable@kernel.org
```

For adding version info, the following comment style should be used:

```
stable@vger.kernel.org # version info
```

COMMIT_COMMENT_SYMBOL

Commit log lines starting with a '#' are ignored by git as comments. To solve this problem addition of a single

space infront of the log line is enough.

COMMIT_MESSAGE

The patch is missing a commit description. A brief description of the changes made by the patch should be added.

See: https://www.kernel.org/doc/html/latest/process/submitting-patches.html#describe-your-changes

EMAIL_SUBJECT

Naming the tool that found the issue is not very useful in the subject line. A good subject line summarizes the change that the patch brings.

See: https://www.kernel.org/doc/html/latest/process/submitting-patches.html#describe-your-changes

FROM_SIGN_OFF_MISMATCH

The author's email does not match with that in the Signed-off-by: line(s). This can be sometimes caused due to an improperly configured email client.

This message is emitted due to any of the following reasons:

```
- The email names do not match.
- The email addresses do not match.
- The email subaddresses do not match.
- The email comments do not match.
```

MISSING_SIGN_OFF

The patch is missing a Signed-off-by line. A signed-off-by line should be added according to Developer's certificate of Origin.

See: https://www.kernel.org/doc/html/latest/process/submitting-patches.html#sign-your-work-the-developer-s-certificate-of-origin

NO_AUTHOR_SIGN_OFF

The author of the patch has not signed off the patch. It is required that a simple sign off line should be present at the end of explanation of the patch to denote that the author has written it or otherwise has the rights to pass it on as an open source patch.

See: https://www.kernel.org/doc/html/latest/process/submitting-patches.html#sign-your-work-the-developer-s-certificate-of-origin

DIFF_IN_COMMIT_MSG

Avoid having diff content in commit message. This causes problems when one tries to apply a file containing both the changelog and the diff because patch(1) tries to apply the diff which it found in the changelog.

See: https://lore.kernel.org/lkml/20150611134006.9df79a893e3636019ad2759e@linux-foundation.org/

GERRIT_CHANGE_ID

To be picked up by gerrit, the footer of the commit message might have a Change-Id like:

```
Change-Id: Ic8aaa0728a43936cd4c6e1ed590e01ba8f0fbf5b
Signed-off-by: A. U. Thor <author@example.com>
```

The Change-Id line must be removed before submitting.

GIT_COMMIT_ID

The proper way to reference a commit id is: commit <12+ chars of sha1> ("<title line>")

An example may be:

```
Commit e21d2170f36602ae2708 ("video: remove unnecessary
platform_set_drvdata()") removed the unnecessary
platform_set_drvdata(), but left the variable "dev" unused,
delete it.
```

See: https://www.kernel.org/doc/html/latest/process/submitting-patches.html#describe-your-changes

Comparison style

ASSIGN_IN_IF

Do not use assignments in if condition. Example:

```
if ((foo = bar(...)) < BAZ) {
```

should be written as:

```
foo = bar(...);
if (foo < BAZ) {
```

BOOL_COMPARISON

Comparisons of A to true and false are better written as A and !A.

See: https://lore.kernel.org/lkml/1365563834.27174.12.camel@joe-AO722/

COMPARISON_TO_NULL

Comparisons to NULL in the form (foo == NULL) or (foo != NULL) are better written as (!foo) and (foo).

CONSTANT_COMPARISON

Comparisons with a constant or upper case identifier on the left side of the test should be avoided.

Indentation and Line Breaks

CODE_INDENT

Code indent should use tabs instead of spaces. Outside of comments, documentation and Kconfig, spaces are never used for indentation.

See: https://www.kernel.org/doc/html/latest/process/coding-style.html#indentation

DEEP_INDENTATION

Indentation with 6 or more tabs usually indicate overly indented code.

It is suggested to refactor excessive indentation of if/else/for/do/while/switch statements.

See: https://lore.kernel.org/lkml/1328311239.21255.24.camel@joe2Laptop/

SWITCH_CASE_INDENT_LEVEL

switch should be at the same indent as case. Example:

```
switch (suffix) {
case 'G':
case 'g':
        mem <<= 30;
        break;
case 'M':
case 'm':
        mem <<= 20;
        break;
case 'K':
case 'k':
        mem <<= 10;
        fallthrough;
default:
        break;
}
```

See: https://www.kernel.org/doc/html/latest/process/coding-style.html#indentation

LONG_LINE

The line has exceeded the specified maximum length. To use a different maximum line length, the --max-line-length=n option may be added while invoking checkpatch.

Earlier, the default line length was 80 columns. Commit bdc48fa11e46 ("checkpatch/coding-style: deprecate 80-column warning") increased the limit to 100 columns. This is not a hard limit either and it's preferable to stay within 80 columns whenever possible.

See: https://www.kernel.org/doc/html/latest/process/coding-style.html#breaking-long-lines-and-strings

LONG_LINE_STRING

A string starts before but extends beyond the maximum line length. To use a different maximum line length, the --max-line-length=n option may be added while invoking checkpatch.

See: https://www.kernel.org/doc/html/latest/process/coding-style.html#breaking-long-lines-and-strings

LONG_LINE_COMMENT

A comment starts before but extends beyond the maximum line length. To use a different maximum line length, the --max-line-length=n option may be added while invoking checkpatch.

See: https://www.kernel.org/doc/html/latest/process/coding-style.html#breaking-long-lines-and-strings

SPLIT_STRING

Quoted strings that appear as messages in userspace and can be grepped, should not be split across multiple

lines.

See: https://lore.kernel.org/lkml/20120203052727.GA15035@leaf/

MULTILINE_DEREFERENCE

A single dereferencing identifier spanned on multiple lines like:

```
struct_identifier->member[index].
member = <foo>;
```

is generally hard to follow. It can easily lead to typos and so makes the code vulnerable to bugs.

If fixing the multiple line dereferencing leads to an 80 column violation, then either rewrite the code in a more simple way or if the starting part of the dereferencing identifier is the same and used at multiple places then store it in a temporary variable, and use that temporary variable only at all the places. For example, if there are two dereferencing identifiers:

```
member1->member2->member3.foo1;
member1->member2->member3.foo2;
```

then store the member1->member2->member3 part in a temporary variable. It not only helps to avoid the 80 column violation but also reduces the program size by removing the unnecessary dereferences.

But if none of the above methods work then ignore the 80 column violation because it is much easier to read a dereferencing identifier on a single line.

TRAILING_STATEMENTS

Trailing statements (for example after any conditional) should be on the next line. Statements, such as:

```
if (x == y) break;
```

should be:

```
if (x == y)
        break;
```

Macros, Attributes and Symbols

ARRAY_SIZE

The ARRAY_SIZE(foo) macro should be preferred over sizeof(foo)/sizeof(foo[0]) for finding number of elements in an array.

The macro is defined in include/linux/kernel.h:

```
#define ARRAY_SIZE(x) (sizeof(x) / sizeof((x)[0]))
```

AVOID_EXTERNS

Function prototypes don't need to be declared extern in .h files. It's assumed by the compiler and is unnecessary.

AVOID_L_PREFIX

Local symbol names that are prefixed with *.L* should be avoided, as this has special meaning for the assembler; a symbol entry will not be emitted into the symbol table. This can prevent *objtool* from generating correct unwind info.

Symbols with STB_LOCAL binding may still be used, and *.L* prefixed local symbol names are still generally usable within a function, but *.L* prefixed local symbol names should not be used to denote the beginning or end of code regions via *SYM_CODE_START_LOCAL*/*SYM_CODE_END*

BIT_MACRO

Defines like: 1 << <digit> could be BIT(digit). The BIT() macro is defined via include/linux/bits.h:

```
#define BIT(nr)         (1UL << (nr))
```

CONST_READ_MOSTLY

When a variable is tagged with the __read_mostly annotation, it is a signal to the compiler that accesses to the variable will be mostly reads and rarely(but NOT never) a write.

const __read_mostly does not make any sense as const data is already read-only. The __read_mostly annotation thus should be removed.

DATE_TIME

It is generally desirable that building the same source code with the same set of tools is reproducible, i.e. the output is always exactly the same.

The kernel does *not* use the `__DATE__` and `__TIME__` macros, and enables warnings if they are used as they can

lead to non-deterministic builds.

See: https://www.kernel.org/doc/html/latest/kbuild/reproducible-builds.html#timestamps

DEFINE_ARCH_HAS

The ARCH_HAS_xyz and ARCH_HAVE_xyz patterns are wrong.

For big conceptual features use Kconfig symbols instead. And for smaller things where we have compatibility fallback functions but want architectures able to override them with optimized ones, we should either use weak functions (appropriate for some cases), or the symbol that protects them should be the same symbol we use.

See: https://lore.kernel.org/lkml/CA+55aFycQ9XJvEOsiM3txHL5bjUc8CeKWJNR_H+MiicaddB42Q@mail.gmail.com/

DO_WHILE_MACRO_WITH_TRAILING_SEMICOLON

do {} while(0) macros should not have a trailing semicolon.

INIT_ATTRIBUTE

Const init definitions should use __initconst instead of __initdata.

Similarly init definitions without const require a separate use of const.

INLINE_LOCATION

The inline keyword should sit between storage class and type.

For example, the following segment:

```
inline static int example_function(void)
{
        ...
}
```

should be:

```
static inline int example_function(void)
{
        ...
}
```

MISPLACED_INIT

It is possible to use section markers on variables in a way which gcc doesn't understand (or at least not the way the developer intended):

```
static struct __initdata samsung_pll_clock exynos4_plls[nr_plls] = {
```

does not put exynos4_plls in the .initdata section. The __initdata marker can be virtually anywhere on the line, except right after "struct". The preferred location is before the "=" sign if there is one, or before the trailing ";" otherwise.

See: https://lore.kernel.org/lkml/1377655732.3619.19.camel@joe-AO722/

MULTISTATEMENT_MACRO_USE_DO_WHILE

Macros with multiple statements should be enclosed in a do - while block. Same should also be the case for macros starting with *if* to avoid logic defects:

```
#define macrofun(a, b, c)                 \
  do {                                    \
          if (a == 5)                     \
                  do_this(b, c);          \
  } while (0)
```

See: https://www.kernel.org/doc/html/latest/process/coding-style.html#macros-enums-and-rtl

PREFER_FALLTHROUGH

Use the *fallthrough;* pseudo keyword instead of /* *fallthrough* */ like comments.

TRAILING_SEMICOLON

Macro definition should not end with a semicolon. The macro invocation style should be consistent with function calls. This can prevent any unexpected code paths:

```
#define MAC do_something;
```

If this macro is used within a if else statement, like:

```
if (some_condition)
        MAC;
```

```
else
        do_something;
```

Then there would be a compilation error, because when the macro is expanded there are two trailing semicolons, so the else branch gets orphaned.

See: https://lore.kernel.org/lkml/1399671106.2912.21.camel@joe-AO725/

SINGLE_STATEMENT_DO_WHILE_MACRO

For the multi-statement macros, it is necessary to use the do-while loop to avoid unpredictable code paths. The do-while loop helps to group the multiple statements into a single one so that a function-like macro can be used as a function only.

But for the single statement macros, it is unnecessary to use the do-while loop. Although the code is syntactically correct but using the do-while loop is redundant. So remove the do-while loop for single statement macros.

WEAK_DECLARATION

Using weak declarations like __attribute__((weak)) or __weak can have unintended link defects. Avoid using them.

Functions and Variables

CAMELCASE

Avoid CamelCase Identifiers.

See: https://www.kernel.org/doc/html/latest/process/coding-style.html#naming

CONST_CONST

Using *const <type> const * * is generally meant to be written *const <type> * * *const*.

CONST_STRUCT

Using const is generally a good idea. Checkpatch reads a list of frequently used structs that are always or almost always constant.

The existing structs list can be viewed from *scripts/const_structs.checkpatch*.

See: https://lore.kernel.org/lkml/alpine.DEB.2.10.1608281509480.3321@hadrien/

EMBEDDED_FUNCTION_NAME

Embedded function names are less appropriate to use as refactoring can cause function renaming. Prefer the use of "%s", __func__ to embedded function names.

Note that this does not work with -f(--file) checkpatch option as it depends on patch context providing the function name.

FUNCTION_ARGUMENTS

This warning is emitted due to any of the following reasons:

1. Arguments for the function declaration do not follow the identifier name. Example:

   ```
   void foo
   (int bar, int baz)
   ```

   This should be corrected to:

   ```
   void foo(int bar, int baz)
   ```

2. Some arguments for the function definition do not have an identifier name. Example:

   ```
   void foo(int)
   ```

   All arguments should have identifier names.

FUNCTION_WITHOUT_ARGS

Function declarations without arguments like:

```
int foo()
```

should be:

```
int foo(void)
```

GLOBAL_INITIALISERS

Global variables should not be initialized explicitly to 0 (or NULL, false, etc.). Your compiler (or rather your loader, which is responsible for zeroing out the relevant sections) automatically does it for you.

INITIALISED_STATIC

Static variables should not be initialized explicitly to zero. Your compiler (or rather your loader) automatically does it for you.

MULTIPLE_ASSIGNMENTS

Multiple assignments on a single line makes the code unnecessarily complicated. So on a single line assign value to a single variable only, this makes the code more readable and helps avoid typos.

RETURN_PARENTHESES

return is not a function and as such doesn't need parentheses:

```
return (bar);
```

can simply be:

```
return bar;
```

Permissions

DEVICE_ATTR_PERMS

The permissions used in DEVICE_ATTR are unusual. Typically only three permissions are used - 0644 (RW), 0444 (RO) and 0200 (WO).

See: https://www.kernel.org/doc/html/latest/filesystems/sysfs.html#attributes

EXECUTE_PERMISSIONS

There is no reason for source files to be executable. The executable bit can be removed safely.

EXPORTED_WORLD_WRITABLE

Exporting world writable sysfs/debugfs files is usually a bad thing. When done arbitrarily they can introduce serious security bugs. In the past, some of the debugfs vulnerabilities would seemingly allow any local user to write arbitrary values into device registers - a situation from which little good can be expected to emerge.

See: https://lore.kernel.org/linux-arm-kernel/cover.1296818921.git.segoon@openwall.com/

NON_OCTAL_PERMISSIONS

Permission bits should use 4 digit octal permissions (like 0700 or 0444). Avoid using any other base like decimal.

SYMBOLIC_PERMS

Permission bits in the octal form are more readable and easier to understand than their symbolic counterparts because many command-line tools use this notation. Experienced kernel developers have been using these traditional Unix permission bits for decades and so they find it easier to understand the octal notation than the symbolic macros. For example, it is harder to read S_IWUSR|S_IRUGO than 0644, which obscures the developer's intent rather than clarifying it.

See: https://lore.kernel.org/lkml/CA+55aFw5v23T-zvDZp-MmD_EYxF8WbafwwB59934FV7g21uMGQ@mail.gmail.com/

Spacing and Brackets

ASSIGNMENT_CONTINUATIONS

Assignment operators should not be written at the start of a line but should follow the operand at the previous line.

BRACES

The placement of braces is stylistically incorrect. The preferred way is to put the opening brace last on the line, and put the closing brace first:

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

This applies for all non-functional blocks. However, there is one special case, namely functions: they have the opening brace at the beginning of the next line, thus:

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

See: https://www.kernel.org/doc/html/latest/process/coding-style.html#placing-braces-and-spaces

BRACKET_SPACE

Whitespace before opening bracket '[' is prohibited. There are some exceptions:

1. With a type on the left:

```
int [] a;
```

2. At the beginning of a line for slice initialisers:

```
[0...10] = 5,
```

3. Inside a curly brace:

```
= { [0...10] = 5 }
```

CONCATENATED_STRING

Concatenated elements should have a space in between. Example:

```
printk(KERN_INFO"bar");
```

should be:

```
printk(KERN_INFO "bar");
```

ELSE_AFTER_BRACE

else { should follow the closing block *}* on the same line.

See: https://www.kernel.org/doc/html/latest/process/coding-style.html#placing-braces-and-spaces

LINE_SPACING

Vertical space is wasted given the limited number of lines an editor window can display when multiple blank lines are used.

See: https://www.kernel.org/doc/html/latest/process/coding-style.html#spaces

OPEN_BRACE

The opening brace should be following the function definitions on the next line. For any non-functional block it should be on the same line as the last construct.

See: https://www.kernel.org/doc/html/latest/process/coding-style.html#placing-braces-and-spaces

POINTER_LOCATION

When using pointer data or a function that returns a pointer type, the preferred use of * is adjacent to the data name or function name and not adjacent to the type name. Examples:

```
char *linux_banner;
unsigned long long memparse(char *ptr, char **retptr);
char *match_strdup(substring_t *s);
```

See: https://www.kernel.org/doc/html/latest/process/coding-style.html#spaces

SPACING

Whitespace style used in the kernel sources is described in kernel docs.

See: https://www.kernel.org/doc/html/latest/process/coding-style.html#spaces

TRAILING_WHITESPACE

Trailing whitespace should always be removed. Some editors highlight the trailing whitespace and cause visual distractions when editing files.

See: https://www.kernel.org/doc/html/latest/process/coding-style.html#spaces

UNNECESSARY_PARENTHESES

Parentheses are not required in the following cases:

1. Function pointer uses:

```
(foo->bar)();
```

could be:

```
foo->bar();
```

2. Comparisons in if:

```
if ((foo->bar) && (foo->baz))
if ((foo == bar))
```

could be:

```
if (foo->bar && foo->baz)
if (foo == bar)
```

3. addressof/dereference single Lvalues:

```
&(foo->bar)
*(foo->bar)
```

could be:

```
&foo->bar
*foo->bar
```

WHILE_AFTER_BRACE

while should follow the closing bracket on the same line:

```
do {
        ...
} while(something);
```

See: https://www.kernel.org/doc/html/latest/process/coding-style.html#placing-braces-and-spaces

Others

CONFIG_DESCRIPTION

Kconfig symbols should have a help text which fully describes it.

CORRUPTED_PATCH

The patch seems to be corrupted or lines are wrapped. Please regenerate the patch file before sending it to the maintainer.

CVS_KEYWORD

Since linux moved to git, the CVS markers are no longer used. So, CVS style keywords (Id, $Revision$, Log) should not be added.

DEFAULT_NO_BREAK

switch default case is sometimes written as "default:;". This can cause new cases added below default to be defective.

A "break;" should be added after empty default statement to avoid unwanted fallthrough.

DOS_LINE_ENDINGS

For DOS-formatted patches, there are extra ^M symbols at the end of the line. These should be removed.

DT_SCHEMA_BINDING_PATCH

DT bindings moved to a json-schema based format instead of freeform text.

See: https://www.kernel.org/doc/html/latest/devicetree/bindings/writing-schema.html

DT_SPLIT_BINDING_PATCH

Devicetree bindings should be their own patch. This is because bindings are logically independent from a driver implementation, they have a different maintainer (even though they often are applied via the same tree), and it makes for a cleaner history in the DT only tree created with git-filter-branch.

See: https://www.kernel.org/doc/html/latest/devicetree/bindings/submitting-patches.html#i-for-patch-submitters

EMBEDDED_FILENAME

Embedding the complete filename path inside the file isn't particularly useful as often the path is moved around and becomes incorrect.

FILE_PATH_CHANGES

Whenever files are added, moved, or deleted, the MAINTAINERS file patterns can be out of sync or outdated.

So MAINTAINERS might need updating in these cases.

MEMSET

The memset use appears to be incorrect. This may be caused due to badly ordered parameters. Please recheck the usage.

NOT_UNIFIED_DIFF

The patch file does not appear to be in unified-diff format. Please regenerate the patch file before sending it to the

maintainer.

PRINTF_0XDECIMAL

Prefixing 0x with decimal output is defective and should be corrected.

SPDX_LICENSE_TAG

The source file is missing or has an improper SPDX identifier tag. The Linux kernel requires the precise SPDX identifier in all source files, and it is thoroughly documented in the kernel docs.

See: https://www.kernel.org/doc/html/latest/process/license-rules.html

TYPO_SPELLING

Some words may have been misspelled. Consider reviewing them.