

Message logging with printk

printk() is one of the most widely known functions in the Linux kernel. It's the standard tool we have for printing messages and usually the most basic way of tracing and debugging. If you're familiar with printf(3) you can tell printk() is based on it, although it has some functional differences:

- printk() messages can specify a log level.
- the format string, while largely compatible with C99, doesn't follow the exact same specification. It has some extensions and a few limitations (no %n or floating point conversion specifiers). See [.ref: How to get printk format specifiers right <printk-specifiers>](#).

System Message: ERROR/3 (D:\onboarding-resources\sample-onboarding-resources\linux-master\Documentation\core-api\[linux-master][Documentation][core-api]printk-basics.rst, line 14); [backlink](#)
Unknown interpreted text role "ref".

All printk() messages are printed to the kernel log buffer, which is a ring buffer exported to userspace through /dev/kmsg. The usual way to read it is using dmesg.

printk() is typically used like this:

```
printk(KERN_INFO "Message: %s\n", arg);
```

where KERN_INFO is the log level (note that it's concatenated to the format string, the log level is not a separate argument). The available log levels are:

| Name | String | Alias function |
|--------------|--------|---|
| KERN_EMERG | "0" | pr_emerg() |
| KERN_ALERT | "1" | pr_alert() |
| KERN_CRIT | "2" | pr_crit() |
| KERN_ERR | "3" | pr_err() |
| KERN_WARNING | "4" | pr_warn() |
| KERN_NOTICE | "5" | pr_notice() |
| KERN_INFO | "6" | pr_info() |
| KERN_DEBUG | "7" | pr_debug() and pr_devel() if DEBUG is defined |
| KERN_DEFAULT | "" | |
| KERN_CONT | "c" | pr_cont() |

The log level specifies the importance of a message. The kernel decides whether to show the message immediately (printing it to the current console) depending on its log level and the current *console_loglevel* (a kernel variable). If the message priority is higher (lower log level value) than the *console_loglevel* the message will be printed to the console.

If the log level is omitted, the message is printed with KERN_DEFAULT level.

You can check the current *console_loglevel* with:

```
$ cat /proc/sys/kernel/printk
4      4      1      7
```

The result shows the *current*, *default*, *minimum* and *boot-time-default* log levels.

To change the current console_loglevel simply write the desired level to /proc/sys/kernel/printk. For example, to print all messages to the console:

```
# echo 8 > /proc/sys/kernel/printk
```

Another way, using dmesg:

```
# dmesg -n 5
```

sets the console_loglevel to print KERN_WARNING (4) or more severe messages to console. See dmesg(1) for more information.

As an alternative to printk() you can use the pr_*() aliases for logging. This family of macros embed the log level in the macro names. For example:

```
pr_info("Info message no. %d\n", msg_num);
```

prints a KERN_INFO message.

Besides being more concise than the equivalent printk() calls, they can use a common definition for the format string through the pr_fmt() macro. For instance, defining this at the top of a source file (before any #include directive):

```
#define pr_fmt(fmt) "%s:%s: " fmt, KBUILD_MODNAME, __func__
```

would prefix every `pr_*`() message in that file with the module and function name that originated the message.

For debugging purposes there are also two conditionally-compiled macros: `pr_debug()` and `pr_devel()`, which are compiled-out unless `DEBUG` (or also `CONFIG_DYNAMIC_DEBUG` in the case of `pr_debug()`) is defined.

Function reference

System Message: ERROR/3 (D:\onboarding-resources\sample-onboarding-resources\linux-master\Documentation\core-api\[linux-master] [Documentation] [core-api] printk-basics.rst, line 110)

Unknown directive type "kernel-doc".

```
.. kernel-doc:: include/linux/printk.h
   :functions: printk pr_emerg pr_alert pr_crit pr_err pr_warn pr_notice pr_info
   :pr_fmt pr_debug pr_devel pr_cont
```