**1. Introduction**

Signals are one of the main inter-process communication (IPC) methods in Linux. Some signals are for killing processes, while others are simply notifications.

In this tutorial, **we explore ways to send a non-terminating signal to a process**. First, we list and discuss interrupting and non-interrupting signals. Next, we describe a way to handle them. Finally, we turn to potentially safe signals and their possible pitfalls.

We tested the code in this tutorial on Debian 11 (Bullseye) with GNU Bash 5.1.4. It should work in most POSIX-compliant environments.

**2. Interrupting and Non-interrupting Signals**

**To get a complete list of every signal from a given system, we can use the *kill* command with its *-l* flag**:

$ kill -l

1) SIGHUP 2) SIGINT 3) SIGQUIT 4) SIGILL 5) SIGTRAP

6) SIGABRT 7) SIGBUS 8) SIGFPE 9) SIGKILL 10) SIGUSR1

11) SIGSEGV 12) SIGUSR2 13) SIGPIPE 14) SIGALRM 15) SIGTERM

16) SIGSTKFLT 17) SIGCHLD 18) SIGCONT 19) SIGSTOP 20) SIGTSTP

21) SIGTTIN 22) SIGTTOU 23) SIGURG 24) SIGXCPU 25) SIGXFSZ

26) SIGVTALRM 27) SIGPROF 28) SIGWINCH 29) SIGIO 30) SIGPWR

31) SIGSYS 34) SIGRTMIN 35) SIGRTMIN+1 36) SIGRTMIN+2 37) SIGRTMIN+3

38) SIGRTMIN+4 39) SIGRTMIN+5 40) SIGRTMIN+6 41) SIGRTMIN+7 42) SIGRTMIN+8

43) SIGRTMIN+9 44) SIGRTMIN+10 45) SIGRTMIN+11 46) SIGRTMIN+12 47) SIGRTMIN+13

48) SIGRTMIN+14 49) SIGRTMIN+15 50) SIGRTMAX-14 51) SIGRTMAX-13 52) SIGRTMAX-12

53) SIGRTMAX-11 54) SIGRTMAX-10 55) SIGRTMAX-9 56) SIGRTMAX-8 57) SIGRTMAX-7

58) SIGRTMAX-6 59) SIGRTMAX-5 60) SIGRTMAX-4 61) SIGRTMAX-3 62) SIGRTMAX-2

63) SIGRTMAX-1 64) SIGRTMAX

Officially, Linux provides most signals that SystemV, Berkeley Software Distribution (BSD), and POSIX support, but there are special cases:

* *SIGEMT* is unsupported
* *SIGINFO* is unsupported
* *SIGSYS* is unsupported
* *SIGABRT* is the same as *SIGIOT*
* *SIGIO*, *SIGPOLL*, and *SIGURG* are the same
* *SIGBUS* is *SIGUNUSED*

**2.1. Ignorable Interrupting Signals**

Now, let’s separate ignorable signals from the signal table, (part of) the default action of which is to stop or terminate the process:

+-------------+-------------------------------------+

| Signal Name | Function |

+-------------+-------------------------------------+

| SIGHUP| terminal hangup |

+-------------+-------------------------------------+

| SIGINT | interruption request |

+-------------+-------------------------------------+

| SIGQUIT | terminate with core dump |

+-------------+-------------------------------------+

| SIGILL | illegal instruction |

+-------------+-------------------------------------+

| SIGTRAP | trace or breakpoint |

+-------------+-------------------------------------+

| SIGABRT | abort |

+-------------+-------------------------------------+

| SIGBUS | bad memory access or bus error |

+-------------+-------------------------------------+

| SIGFPE | floating-point exception |

+-------------+-------------------------------------+

| SIGUSR1 | custom user signal 1 |

+-------------+-------------------------------------+

| SIGSEGV | invalid memory reference |

+-------------+-------------------------------------+

| SIGUSR2 | custom user signal 2 |

+-------------+-------------------------------------+

| SIGPIPE | write to pipe without readers |

+-------------+-------------------------------------+

| SIGALRM | timer signal |

+-------------+-------------------------------------+

| SIGTERM | like SIGQUIT without a core dump |

+-------------+-------------------------------------+

| SIGSTKFLT | stack fault on coprocessor |

+-------------+-------------------------------------+

| SIGTSTP | stop at terminal |

+-------------+-------------------------------------+

| SIGTTIN | terminal input, background process |

+-------------+-------------------------------------+

| SIGTTOU | terminal output, background process |

+-------------+-------------------------------------+

| SIGURG | urgent condition on socket |

+-------------+-------------------------------------+

| SIGXCPU | exceeded CPU time limit |

+-------------+-------------------------------------+

| SIGXFSZ | exceeded file size limit |

+-------------+-------------------------------------+

| SIGVTALRM | virtual alarm clock |

+-------------+-------------------------------------+

| SIGPROF | profiling timer expired |

+-------------+-------------------------------------+

| SIGIO | input or output possible |

+-------------+-------------------------------------+

| SIGPWR | power failure |

+-------------+-------------------------------------+

| SIGSYS | bad system call |

+-------------+-------------------------------------+

| SIGRT\* | real-time signals |

+-------------+-------------------------------------+Copy

Notably, although *SIGUSR1* and *SIGUSR2* are user-defined, their default action is still to kill the process. Still, all of the above can be ignored.

**2.2. Non-ignorable Interrupting Signals**

From the interrupting signals, there is a subset we’re unable to ignore:

* *SIGKILL* – terminate process unconditionally
* *SIGSTOP* – halt and place process in the background

Of course, even *SIGKILL* might not always work, but it’s the last resort.

**2.3. Non-interrupting Signals**

Finally, we can turn to signals that we can ignore but wouldn’t terminate the process if we don’t handle the following:

* *SIGCHLD* – sent by the kernel when a child process completes
* *SIGCLD* – synonym for *SIGCHLD*
* *SIGCONT* – continue if stopped
* *SIGURG* – urgent condition on socket
* *SIGWINCH* – window changed

Armed with this knowledge, we can check ways to send a signal without killing a process.

**3. Trap to Prevent Termination**

In Linux, **we can use the *trap* command to intercept signals so that we can handle them**:

$ trap 'echo "SIGINT"' SIGINT

$ kill -SIGINT $$

SIGINT

$Copy

In this case, we set a handler for the *SIGINT* signal in which we just *echo* the signal name. After that, we send the signal by passing *kill* the shell process ID (PID) *$$*.

Furthermore, using this method, **we can set and invoke handlers for any and all available signals (here, *64*)**:

$ kill -SIGCONT $$

$ trap 'echo "TRAP"' $(seq 1 64)

$ kill -SIGCONT $$

TRAP

$Copy

In addition, **we can even prevent the default action for all signals except *SIGKILL* and *SIGSTOP***:

$ trap 'echo "TRAP"' $(seq 1 64)

$ kill -SIGILL $$

TRAP

$ kill -SIGUSR1 $$

TRAP

$ kill -SIGKILL $$

KilledCopy

In this case, **we invoke several signals that usually terminate a process, but only *SIGKILL* actually manages to do that, as we trap the rest**.

**4. Safe Signals**

Although receiving a specific signal without having a trap or handler might not make much sense, **we can still leverage signals that don’t stop or terminate our process by default**:

$ kill -SIGCONT $$

$Copy

In theory, *SIGCONT* is harmless, as it only performs its task if a process is halted. Otherwise, it should do nothing.

However, this method doesn’t come without its pitfalls:

$ kill -SIGINT $$

$ kill -SIGTERM $$

$Copy

Normally, *SIGINT* and *SIGTERM* terminate processes if they aren’t handled. Yet, sending them to the shell doesn’t kill it in this case. The reason for that is because **some applications may have internal handlers for the signals Linux IPC mechanism**.

Still, this may mislead us into thinking *SIGINT* and *SIGTERM* are non-terminating in general, which they aren’t:

$ sleep 10 &

[1] 666

$ kill -SIGTERM 666

$

[1]+ Terminated sleep 10Copy

In fact, this goes both ways – **we can choose a seemingly safe signal, which can still terminate the process that receives it**. Thus, we should thread carefully when trapping or sending any signal.

Still, we can employ this method with the correct signals when debugging with tools like *gdb* or just debugging Bash scripts in general.