$ cd reference / Signals and exit codes

Unix signals and exit codes

An exit code above 128 usually means a signal ended the process. This page lists the signals you meet in practice, the exit codes shells and tools use, and the order in which service managers send them.

[01] Common signals

NO.	NAME	DEFAULT ACTION	USUALLY SENT BY
1	SIGHUP	terminate	the terminal closing; many daemons reload their configuration on it instead
2	SIGINT	terminate	Ctrl+C in the terminal
3	SIGQUIT	core dump	Ctrl+\ in the terminal; the JVM prints a thread dump on it
4	SIGILL	core dump	an illegal CPU instruction, for example a binary built for a newer CPU
6	SIGABRT	core dump	the program itself, through abort(), often after a failed assertion
7	SIGBUS	core dump	a bad memory access, such as reading a memory-mapped file that was truncated
8	SIGFPE	core dump	an arithmetic error such as integer division by zero
9	SIGKILL	terminate	`kill -9`, the out-of-memory killer, a service manager whose stop timeout ran out; cannot be caught, blocked or ignored
10	SIGUSR1	terminate	an administrator or another program; the meaning is defined by the application
11	SIGSEGV	core dump	an invalid memory reference: a crash in native code
12	SIGUSR2	terminate	as SIGUSR1
13	SIGPIPE	terminate	writing to a pipe or socket whose reader has gone, as in `producer | head`
14	SIGALRM	terminate	a timer set with alarm()
15	SIGTERM	terminate	`kill` without options, systemd, Docker and Kubernetes when stopping: the polite request to exit
17	SIGCHLD	ignore	the kernel, when a child process stops or exits

NO.	NAME	DEFAULT ACTION	USUALLY SENT BY
18	SIGCONT	continue	`fg`, `bg`, `kill -CONT`
19	SIGSTOP	stop	`kill -STOP`; cannot be caught, blocked or ignored
20	SIGTSTP	stop	Ctrl+Z in the terminal
24	SIGXCPU	core dump	the CPU time limit (`ulimit -t`) running out
25	SIGXFSZ	core dump	writing past the file size limit (`ulimit -f`)
28	SIGWINCH	ignore	the terminal window changing size

The numbers are those of Linux on x86 and ARM, the same on most other architectures. Alpha, SPARC and MIPS number several signals differently, so scripts should use names: `kill -TERM`, not `kill -15`. "Core dump" means the process terminates and the kernel writes a core file if the core size limit allows it (`ulimit -c`).

[02] Sending and handling signals

TASK	COMMAND
Ask a process to exit	`kill PID`
Send a specific signal by name	`kill -HUP PID`
Check that a process exists and you may signal it	`kill -0 PID`
Signal every process named exactly app	`pkill -TERM -x app`
Signal a whole process group	`kill -TERM -- -PGID`
Signal a service and all its processes (systemd)	`sudo systemctl kill -s HUP app.service`
List signal names and numbers	`kill -l`
Clean up when a script exits or is interrupted	`trap 'rm -f "$tmp"' EXIT`
Give a command 30 seconds, then TERM, then KILL 5 seconds later	`timeout -k 5 30 ./job.sh`

Send SIGTERM first and SIGKILL only if the process does not exit. A process killed with SIGKILL gets no chance to flush buffers, remove lock files or finish a transaction.

[03] Exit codes

CODE	MEANING
0	success
1	general failure; most programs use it for any error
2	incorrect usage of a shell builtin in bash; many programs also use it for bad arguments, and `grep` for errors (1 means "no match")
124	`timeout` stopped the command because the time ran out
125	`timeout` itself failed
126	the command was found but could not be executed: no execute permission, or not a valid executable
127	command not found: a typo, or a PATH that does not include it, as under cron
128 + n	the command was ended by signal n
130	128 + 2: SIGINT, Ctrl+C
137	128 + 9: SIGKILL; often the out-of-memory killer, or a stop timeout that ran out
143	128 + 15: SIGTERM; a normal stop request that the program did not handle
255	`exit -1` and other out-of-range values; `ssh` also returns 255 for its own errors

An exit status is 8 bits wide. `exit 256` gives 0 and `exit -1` gives 255, so never use a count of errors as the exit status. The 128 + n values are what the shell reports; the process itself did not exit. Python's `subprocess` reports the same case as a negative return code, -15 for SIGTERM.

```
./job.sh; echo $?          # status of the last command
false | true; echo $?       # 0: a pipeline returns the last command's status
set -o pipefail             # make a pipeline fail if any command in it fails
```

[04] sysexits.h codes

Some programs, mail software in particular, use the BSD codes from `sysexits.h`. They are a convention, not a requirement of POSIX.

CODE	NAME	MEANING
64	EX_USAGE	the command was used incorrectly
65	EX_DATAERR	the input data was incorrect
66	EX_NOINPUT	an input file did not exist or was not readable

CODE	NAME	MEANING
67	`EX_NOUSER`	the user does not exist
68	`EX_NOHOST`	the host does not exist
69	`EX_UNAVAILABLE`	a service is unavailable
70	`EX_SOFTWARE`	internal software error
71	`EX_OSERR`	operating system error, such as a failed fork
72	`EX_OSFILE`	a system file is missing or invalid
73	`EX_CANTCREAT`	an output file cannot be created
74	`EX_IOERR`	input or output error
75	`EX_TEMPFAIL`	temporary failure; trying again later may work
76	`EX_PROTOCOL`	the remote side broke the protocol
77	`EX_NOPERM`	permission denied
78	`EX_CONFIG`	configuration error

[05] How services are stopped

MANAGER	FIRST	THEN	WAITS BY DEFAULT
systemd	`KillSignal=`, default SIGTERM	SIGKILL to what is left	`TimeoutStopSec=`, 90 s unless changed
Docker (`docker stop`)	SIGTERM, or the image's `STOPSIGNAL`	SIGKILL	10 s for Linux containers
Kubernetes	SIGTERM (after any preStop hook)	SIGKILL	`terminationGracePeriodSeconds`, 30 s
`timeout`	SIGTERM, or the signal from `-s`	SIGKILL only with `-k`	the duration you give

A process that takes the full grace period to stop is usually ignoring SIGTERM. In a container this happens often for a reason that is easy to miss: the first process in a PID namespace receives only the signals it has installed a handler for, so a program without a SIGTERM handler running as PID 1 ignores the default action and waits to be killed. Run it under a small init process, such as `docker run --init`, or handle SIGTERM in the program.

[06] Sources

[1] signal(7): numbers, default actions, architectures

[2] pid_namespaces(7): signals to the init process

[3] Bash manual: exit status

[4] timeout(1): exit status

[5] sysexits.h(3head)

[6] systemd.kill(5): KillSignal=

[7] systemd-system.conf(5): DefaultTimeoutStopSec=

[8] Docker: docker container stop

[9] Kubernetes: termination of pods

[10] Python: Popen.returncode