Signal and Trap in Unix / Linux

Signals are software interrupt signals that are sent to a program that indicates a serious event has occurred. These events can be very diverse from user requests to illegal memory access. Some signals, like signal interrupts, indicate that a user has asked the program to do something without being in control.

Signals are software interrupt signals that are sent to a program that indicates a serious event has occurred. These events can be very diverse from user requests to illegal memory access. Some signals, like signal interrupts, indicate that a user has asked the program to do something without being in control.

The following table lists the common signals that you may encounter or want to use it in your programs:

Signal name Signal signal number Description SIGHUP 1 Delay the check on terminal management or stop of process management. SIGINT 2 Notified if the user sends an interrupt signal (Ctrl + C). SIGQUIT 3 Notified if the user sends a quit signal (Ctrl + D). SIGFPE 8 Be notified if an illegal math activity is tried. SIGKILL 9 If a process receives this signal, it must exit immediately and will not perform cleaning operations. SIGALRM 14 Signal signaling number of times (Alarm Clock). SIGTERM 15 Software end signal (sent by sigkill by default).

List of signals in Unix / Linux

There is an easy way to list all signals supported by your system. Just notify **kill -l** command and it will display all supported **signals** .

```
$ kill - l 1 ) SIGHUP 2 ) SIGINT 3 ) SIGQUIT 4 ) SIGILL 5 ) SIGTRAP 6 ) SIGABR
```

The actual list of signals is diverse and different between Solaris, HP-UX and Linux.

Default activities in Unix / Linux

Each signal has a default operation associated with it. The default operation with a **signal** is the operation that a script or program executes when it receives a signal.

One of the default actions can be:

End the process

Skip signal

Core memory dump. It creates a file called the core containing the memory image of the process when it receives the signal

Stop the process

Continue the process to stop

Send signals in Unix / Linux

There are several methods for sending signals to a program or a script. One of the most common methods is for the user to press Ctrl + C or the stop key while a script is running.

When you press the **Ctrl + C key** , a SIGINT is sent to the script and when each specified action ends the script.

Another common method to send a signal is to use **a kill command** that has the following syntax:

```
$ kill - signal pid
```

Here, the signal is either the number or the name of the signal to send and pid is the process ID that the signal should be sent to. For example:

```
$ kill - 1 1001
```

Send the HUP or signal to a program that is running with the process ID of 1001. To send a kill signal to the same process, use the following command:

```
$ kill - 9 1001
```

It will cancel the running process with process ID 1001.

Trap signal in Unix / Linux

When you press **Ctrl + C** or the stop key at your terminal during the process of running a shell program, the program is usually terminated immediately, and the command prompt appears again. This may not always interest you. For example, you might leave a bunch of temporary files that aren't cleaned up.

Trap signal is quite easy, and the trap command has the following syntax:

```
$ trap commands signals
```

Here, the **command** can be any valid Unix command, or even a user defined function, and the signal may be the list of signals you want to trap.

There are 3 common uses to trap in shell scripts:

1. Clean up temporary files
2. Skip the signal

Clean up temporary files in Unix / Linux

As an example of the trap command, below shows how you can remove some files and then exit if someone tries to uninstall the program from the terminal.

```
$ trap "rm -f $ WORKDIR / work1 $$ $ WORKDIR / dataout $$; exit" 2
```

From the point in the shell program that this trap is executed, work1 $$ and dataout $$ files will be automatically removed if the signal number 2 is received by the program.

So if the user interrupts the execution of the program then after this trap is run, you can be sure that these two files will be cleaned up. The exit command followed by rm is required because without it, the program will continue to run at the point where it stops when the signal is received.

Signal No. 1 was created to delay: or someone intentionally hung the line or the line was randomly disconnected.

You can edit the trap first to also remove the two specified files in this case by adding signal number 1 to the signal list.

```
$ trap "rm $ WORKDIR / work1 $$ $ WORKDIR / dataout $$; exit" 1 2
```

Now these files will be removed if the line is suspended or if the Ctrl + C key is pressed.

Commands defined to trap must be surrounded in citations if they contain more than one command. You also notice that the command line shell scans at the time that the trap command is run and also performs a scan again when one of the listed signals is received.

So in the previous example, the value of WORKDIR and $$ will be changed at the time the trap is run. If you want this change to occur at the time that the signal number 1 or 2 is received, you can place the instructions inside the quote as follows:

```
$ trap 'rm $ WORKDIR / work1 $$ $ WORKDIR / dataout $$;  exit ' 1 2
```

Ignore the signals in Unix / Linux

If the command is listed for a trap as null, then the specified signal will be ignored when received. For example, the following command:

```
$ trap '' 2
```

Determine that the interrupt signal is ignored. You may want to ignore certain signals when performing some operations without wanting to be disconnected. You can identify many signals that are ignored as follows:

```
$ trap '' 1 2 3 15
```

Keep in mind that the first argument must be specified for a signal to be ignored and the above is unbalanced with the following, but the latter has its own meaning:

```
$ trap 2
```

If you ignore a signal, all the extra shells ignore that signal as well. However, if you specify an action to be performed when you receive the signal, all sub-shells will still perform the action when receiving the signal.

Reset trap in Unix / Linux

After you have changed the default actions when you receive a signal, you can change it again with the trap, if you simply ignore the first argument as follows:

```
$ trap 1 2
```

It resets the action performed when receiving signal number 1 or 2 back to default mode.

According to Tutorialspoint

Previous article: System log in Unix / Linux

Next article: Useful commands in Unix / Linux