Specify the process by name, which is specified by users who own it, by using:

```
$ killall -u USERNAME process_name
```

If you want `killall` to interactively confirm before killing processes, use the `-i` argument.

The `pkill` command is similar to the `kill` command but it, by default, accepts a process name instead of a process ID. For example:

```
$ pkill process_name
$ pkill -s SIGNAL process_name
```

`SIGNAL` is the signal number. The `SIGNAL` name is not supported with `pkill`. It provides many of the same options that the `kill` command does. Check the `pkill` man pages for more details.

Capturing and responding to signals

`trap` is a command used to assign signal handler to signals in a script. Once a function is assigned to a signal using the `trap` command, while the script runs and it receives a signal, this function is executed upon reception of a corresponding signal.

The syntax is as follows:

```
trap 'signal_handler_function_name' SIGNAL LIST
```

`SIGNAL LIST` is delimited by space. It can be a signal number or a signal name.

Let's write a shell script that responds to the `SIGINT` signal:

```
#/bin/bash
#Filename: sighandle.sh
#Description: Signal handler

function handler()
{
  echo Hey, received signal : SIGINT
}

echo My process ID is $$
# $$ is a special variable that returns process ID of current process/
script
trap 'handler' SIGINT
#handler is the name of the signal handler function for SIGINT signal

while true;
do
  sleep 1
done
```

Run this script in a terminal. When the script is running, if you press *Ctrl + C*, it will show the message by executing the signal handler associated with it. *Ctrl + C* corresponds to a `SIGINT` signal.

The `while` loop is used to keep the process running in an infinite loop without getting terminated. This is done so that it can respond to the signals that are sent to the process asynchronously by another process. The loop that is used to keep the process alive infinitely is often called the **event loop**.

We can send a signal to the script by using the `kill` command and the process ID of the script:

```
$ kill -s SIGINT PROCESS_ID
```

`PROCESS_ID` of the preceding script will be printed when it is executed, or you can find it out by using the `ps` command.

If no signal handlers are specified for signals, it will call the default signal handlers assigned by the operating system. Generally, pressing *Ctrl + C* will terminate a program, as the default handler provided by the operating system will terminate the process. But the custom handler defined here specifies a custom action upon receipt of the signal.

We can define signal handlers for any signals available (`kill -l`) by using the `trap` command. It is also possible to set a single signal handler for multiple signals.

Sending messages to user terminals

A system administrator may need to send messages to the terminal of every user or a specified user on all the machines over a network. This recipe is a guide on how to perform this task.

Getting ready

`wall` is a command that is used to write messages on the terminals of all logged in users. It can be used to convey messages to all logged in users on a server or multiple access machines. Sending messages to all users may not be useful at all times. Terminals are treated as devices in a Linux system and hence, these opened terminals will have a corresponding device node file at `/dev/pts/`. Writing data to a specific device will display messages on the corresponding terminal.