10.2.8 Process Termination Facts

Normally, users should use the exit function (which is coded into nearly all programs) to end a running process. For example, the **Ctrl+c** key sequence is used to exit the top program. However, processes occasionally hang and refuse to exit properly.

This lesson covers the following topics:

- Terminate a running process
- Keep running a process after logging out

Terminate a Running Process

The following commands can be used to terminate a running process.

Command	Function	Example
kill	Terminates a process using a process ID (PID) number and a specific kill signal. Kill signals can be sent using a term or a numeric value. Options include the following: - **-SIGHUP**, **-1** tells the process to shut down and restart. When it restarts, the process will have the same PID it had when the kill signal was sent. - **-SIGINT**, **-2** stops a process as if the Ctrl+c key combination had been used. This option is recommended as the first choice when a process will not stop with its exit function or init script. - **-SIGKILL**, **-9** forces a process to stop when it is unresponsive to other options for exiting or killing it. Running this option does not give the process a chance to clean up any resources that it is using, such as memory. Resources allocated to the process usually remain allocated to it until it is restarted. This option should only be used as a last resort. - **-SIGTERM**, **-15** stops the process cleanly by giving it a chance to release the resources allocated to it. This is the default signal used by the kill command if no signal is specified. This option can be tried if the **-2** option fails to kill the process. - **-l** lists all of the signals that are available for the **kill** command.	**kill -1 6754** shuts down and restarts the process. **kill -9 6754** forces the unresponsive process to stop running. **kill -SIGKILL 6754** (which has the same effect as using **-9**) forces the unresponsive process to stop running. **kill 6754** stops the process with PID 6754 using the default **SIGTERM** signal**.**
killall	Terminates processes the same way as the **kill** command, using their command name instead of their PID. This command uses the same signal commands that **kill** uses.	**killall atd** kills all processes named atd. **killall -9 atd** uses a hard kill to stop the atd process.
pkill	Searches for processes that match the search criteria specified and then sends them a kill signal.	**pkill -SIGTERM -f top** searches through all of the running processes for those with "top" somewhere in their name and sends them the SIGTERM signal.

Keep Running a Process After Logging Out

In addition to killing a running process, you may encounter situations where a process must be kept running even after the user has logged out of the system. Normally, when a user logs out of the terminal session, Linux sends a SIGHUP signal to all the programs associated with that shell session. In response, each process will respond as it is programmed to when it receives a SIGHUP signal. However, a process can be told to ignore SIGHUP signals, which will allow it to remain running even if the user logs out of the shell session where it was started.

Command	Funciton	Example
nohup &	Allows a command or shell script to continue running in the background after logging out from a shell. **nohup** does not automatically put the command it runs in the background; use the ampersand (&) symbol to start a process in the background.	**nohup gedit &** starts the gedit process in the background and leaves it running after logging out of the shell.
screen	Uses multiple shell windows from within a single SSH session. Using **screen**, you can: - Keep processes running while you access the shell prompt through an SSH connection to enter additional commands. - Keep an SSH shell active even if the network connection is closed or goes down.	

- Disconnect and reconnect to a shell session from multiple locations without having to stop and then restart whatever processes were running.

The screen package must be installed on the system before it can be used. After screen is installed, launch it by entering **screen** at the shell prompt. A shell prompt is displayed within a window inside the screen. Each window functions like a normal shell session. Within screen, pressing **Ctrl+a** causes whatever is typed by the user to be sent to the screen process instead of to the shell:

- Pressing **Ctrl+a ?** causes the screen help to be displayed.
- Pressing **Ctrl+a c** causes a new screen window to be created. The old window remains active along with any processes that were running within it.
- Pressing **Ctrl+a n** toggles between open windows in screen.
- Pressing **Ctrl+a d** detaches the screen window and returns the user to the original shell prompt. Whatever was running in the window remains running. In fact, the user can completely log out, and everything will keep running within the detached window.
- Entering **screen -r** reattaches a detached screen window. If multiple detached screen windows exit, the user will be prompted to specify which one to reattach to.