Running jobs in Background

A multitasking system lets a user do more than one job at a time. Since there can be only one job in foreground, the rest of the jobs have to run in the background. There are two ways of doing this: with the shell's **& operator** and **nohup** command. The latter permits to log out while the jobs are running, but the former doesn't allow that.

$ **sort –o emp.lst &**
550

The shell immediately returns a number the PID of the invoked command (550). The prompt is returned and the shell is ready to accept another command even though the previous command has not been terminated yet. The shell however remains the parent of the background process. Using an & many jobs can be run in background as the system load permits.

In the above case, if the shell which has started the background job is terminated, the background job will also be terminated. **nohup** is a command for running a job in background in which case the background job will not be terminated if the shell is close. nohup stands for no hang up.

e.g.
$ **nohup sort-o emp.lst &**
586

The shell returns the PID too. When the **nohup** command is run it sends the standard

output of the command to the file **nohup.out**. Now the user can log out of the system without aborting the command.

JOB CONTROL

1. ps: **ps** is a command for listing processes. Every process in a system will have unique id called process id or PID. This command when used displays the process attributes.
$ **ps**

PID	TTY	TIME	CMD
291	console	0:00	bash

This command shows the PID, the terminal TTY with which the process is associated, the cumulative processor time that has been consumed since the process has started and the process name (CMD).

2. kill: This command sends a signal usually with the intention of killing one or more process. This command is an internal command in most shells. The command uses one or more PIDs as its arguments and by default sends the SIGTERM(15) signal. Thus:
$ **kill 105**
terminates the job having PID 105. The command can take many PIDs at a time to be terminated.

3. sleep: This command makes the calling process sleep until the specified number of seconds or a signal arrives which is not ignored.
$ **sleep** 2

Exercise:

1. Try the following, which illustrates the usage of **ps**:
   $ **(sleep 10; echo done) &**
   $ **ps**

2. Try the following, which illustrates the usage of **kill**:
   $ **(sleep 10; echo done) &**
   $ **kill pid** …..pid is the process id of background process

3. Try the following, which illustrates the usage of **wait**:
   $ **(sleep 10; echo done 1) &**
   $ **(sleep 10; echo done 2) &**
   $ **echo done 3; wait ; echo done 4** ….wait for children

NOTE: The following two utilities and one built-in command allow the listing Controlling the current processes.
ps: generates a list of processes and their attributes, including their names, process ID numbers, controlling terminals, and owners
kill: allows to terminate a process on the basis of its ID number
wait: allows a shell to wait for one or all of its child processes to terminate