Job and Process Control

Job and Process Control

Every time you type a command at a shell, it creates a process. Many processes can be running at the same time, and the operating system determines how they share various resources such as CPU time or memory. Each process is assigned a unique number called a **process ID** or **PID**.

The ps command

The **ps** command can be used to display information about the processes running on the system, including their PIDs and associated commands. Without any arguments, the **ps** command will only display the processes started by you in the current shell:

```
joeuser@athena$ ps
```

PID	TTY	TIME	CMD
10709	pts/0	00:00:00	bash
28622	pts/0	00:00:05	gedit
30116	pts/0	00:00:00	ps

In the example above, the PID column lists the process ID, the TTY column lists the "terminal" (tty) or "psuedo-terminal slave" (pts), the TIME column lists the cumulative processor time used by the process (keep in mind that 1 second is a large value for processor time, which is more often measured in milliseconds), and the CMD column lists the command associated with the process.

As you can see, the **ps** command itself is included in the listing. Each process listing will also include the shell (bash, in this case). In the example above, there's also one additional process, gedit (the GNOME Text Editor).

The **ps** command can display much more information than what appears above. A command option is the **-f** to specify "full format", including the owner of the process and the full command line:

```
joeuser@athena$ ps -f
```

UID	PID	PPID	C	STIME	TTY	TIME	CMD
joeuser	10709	2228	0	11:38	pts/0	00:00:00	bash
joeuser	28622	10709	0	11:40	pts/0	00:00:05	gedit
joeuser	30116	10709	0	11:51	pts/0	00:00:00	ps

The additional columns are: username (UID), Parent Process ID (PPID), percentage current CPU utilization (C), and the current time or date when the process was started (STIME).

Note that by default, the **ps** command only shows processes in the current shell. Other processes (programs you run from the "Applications" menu, for example) will not be listed, unless you tell **ps** to show all processes started by you. You can use the **-u** option to specify that **ps** should display processes for a given user (the $USER environment variable contains your username):

```
joeuser@athena$ ps -f -u $USER
```

UID	PID	PPID	C	STIME	TTY	TIME	CMD
joeuser	2230	2228	0	11:20	?	00:00:10	/usr/bin/metacity --replace
joeuser	2230	2228	0	11:20	?	00:00:10	gnome-panel
joeuser	2235	2228	0	11:20	?	00:00:10	nautilus
joeuser	2342	1	0	11:21	?	00:00:10	gnome-screensaver
joeuser	2344	1	0	11:21	?	00:00:00	/usr/lib/evolution/evolution-data-server
joeuser	10709	2228	0	11:38	pts/0	00:00:00	bash

joeuser	28622	10709	0	11:40	pts/0	00:00:05	gedit
joeuser	30116	10709	0	11:51	pts/0	00:00:00	ps

Note that some of the processes do not have associated terminals, often because they were started automatically as part of the login process. The GNOME environment has a number of processes associated with it (e.g. gnome-screensaver, nautilus) that are started automatically. You should not attempt to kill or modify these processes unless you know what you're doing.

Showing All Processes

There are many more processes running on the system, some run by users such as the "root" user (administrator) or other users. You can view all processes on the system with the **-e** option. Combined with the **-f** option, this can be used to obtain a detailed listing of all processes on the system:

```
joeuser@athena$ ps -ef
```

UID	PID	PPID	C	STIME	TTY	TIME	CMD
root	1	0	0	Jan03	?	00:00:05	/sbin/init
root	2	0	0	Jan03	?	00:00:00	[kthreadd]
root	3	2	0	Jan03	?	00:00:00	[migration/0]
(Listing trimmed for publication)							
joeuser	10709	2228	0	11:38	pts/0	00:00:00	bash

Killing Processes

Normally, processes exit when they have completed on their own or on request from the user (i.e. when someone chooses "Quit" or "Exit" in an application). Occasionally, however, you may need to kill a process that won't exit (i.e. because you gave it the wrong input and it will take too long to finish). The **kill** command is used to kill processes. In its simplest format, you simply specify the PID of the process you wish to kill:

> *joeuser@athena$* ***kill PID***

That causes the operating system to send a **signal** to the process number you specify. There are a number of signals that can be sent to a process, but by default the terminate (TERM) signal is used.

Another common signal is the "hangup" (HUP) signal. In the old days, this signal would be sent when a remote terminal hang up (disconnected the phone or serial line). Some programs interpret the HUP signal as a kind of "reset" signal and restart themselves or reset themselves.

It is possible for programs to ignore or block the TERM or HUP signals. This is sometimes desirable when a program is performing a complicated process and interruption would result in data corruption. If you wish to kill a program in that situation, you can use the KILL signal as a last resort, which will terminate a program immediately, and you will almost certainly lose any unsaved work in the program.

To send a signal other than TERM, you specify it as an option to the kill command:

> *joeuser@athena$* ***kill -HUP PID***
> *joeuser@athena$* ***kill -KILL PID***

Signals have numerical representations as well as abbreviations, and you'll frequently see the KILL signal represented as the number 9. For example, **kill -9 5432** would send the KILL signal to process 5432.

Killing processes by name

It can become time consuming to run **ps**, look for the PID in question, and kill it. Most modern operating systems have the **pkill** command, which behaves like **kill** but takes the name of a process instead of a process ID number. The downside is that this command will kill all processes with that name, and typos can have unintended consequences. Like **kill**, **pkill** can take signals as arguments. For example:

> *joeuser@athena$* ***pkill gnome-terminal***
> *joeuser@athena$* ***pkill -KILL firefox***

Process Ownership

As implied by the fact that processes have associated UIDs, you can only kill processes associated with your UID. If you try and kill a process owned by another user, you will receive an error:

>

```
joeuser@athena$ kill 1
bash: kill: (1) - Operation not permitted
```

Background vs Foreground

When you type a command at your shell prompt, the command is run, and when it completes, you will get another shell prompt. This is called running a command in the **foreground**. Many commands take less than 1 second to complete, so waiting for the command to complete is no problem. Some commands open a new window, and in those cases, you will want to use your Terminal window and the new program simultaneously. In those cases, you can run the command in the **background** by typing an ampersand (**&**) at the end of the line before running the command. For example, the first command below would launch a clock on your desktop, but you would not get a new shell prompt until you closed the clock window. The second command will launch a clock on your desktop and immediately give you a new shell prompt:

```
joeuser@athena$ xclock
joeuser@athena$ xclock &
```

If you have already launched a process in the foreground and want to move it to the background, you can press **Ctrl-z** to suspend the process. Once suspended, you can type **bg** to run it in the background:

```
joeuser@athena$ xclock
<Press Ctrl-z>
joeuser@athena$ bg
[1]+ xclock &
joeuser@athena$
```

The **jobs** command can be used to show you the jobs currently running in the shell, along with the job number. (Note: Job numbers are assigned sequentially and are different from process IDs. Every job has a process ID, not every process has a job number.). Continuing from the previous example:

```
joeuser@athena$ jobs
[1]+ Running xclock &
joeuser@athena$
```

You can bring a background job to the foreground by using the **fg** command. Both the **bg** and **fg** commands will change the most recent job. If you have multiple jobs and wish to bring a specific one to the foreground or background, you can specify the job number (NOT the process ID) with a percent sign. For example:

```
joeuser@athena$ jobs
[1]+ Running xclock &
[2]+ Running gedit &
joeuser@athena$ fg %1
```

would bring job 1 (xclock) into the foreground again.