8 commands for efficient management of Linux processes

There are three main states in the life cycle of an application process: Start, run, and stop. If you want to be a competent Linux administrator, you need to know how to manage each state carefully. The following 8 commands can be used to help you manage the entire process lifecycle.

If you want to be a competent administrator, you need to know how to manage the state of your processes

Start the process

The easiest way to start a process is to type its name on the command line, and then press **Enter.** If you want to start the nginx web server, enter **nginx.** You will probably want to see its version.

```
alan@workstation:~ $nginx alan@workstation:~ $nginx -v nginx version:nginx/1.14.
```

See execution path

The boot process demo above assumes that the executable is in your executable path. Understanding this path is the key to starting and managing processes reliably. Administrators often customize this path according to the purpose they want. You can use **echo $ path** to see the execution path

```
alan@workstation:~ $echo $path /usr/local/sbin:/usr/local/bin:/usr/sbin:/usr/bin
```

which

Use the which command to see the full path of the executable.

```
alan@workstation:~ $which nginx /opt/nginx/bin/nginx
```

We will use the popular web server software nginx as an example. Let's say nginx is installed. If you execute the command **which nginx** and nothing is returned, then nginx cannot be found, since the command only looks for the executable path you specified. There are 3 ways to overcome the situation in which a process cannot be started by name. The first is to enter the full path:

```
alan@workstation:~ $/home/alan/web/prod/nginx/sbin/nginx -v nginx version:nginx/
```

The second solution is to install the application in a directory under the executable path, however, this can sometimes fail, especially if you don't have root access.

The third solution is to update your executable path environment variable, including the installation directory for the specific application you want to use. This solution has to do with the shell. For example, the bash user needs to edit the line **path =** in the .bashrc file.

```
path="$home/web/prod/nginx/sbin:$path"
```

Now, repeat the commands echo and which or try to check the version.

```
alan@workstation:~ $echo $path /home/alan/web/prod/nginx/sbin:/usr/local/sbin:/us
```

Continue to run the process

nohup

When you log out or close Terminal, the process may not continue. In this particular case, you can keep the process running by placing the **nohup** command before the command you want to run. Also, add a trailing mark at the end and the process will be run in the background, allowing you to continue using Terminal. For example you want to run **myprogram.sh.**

```
nohup myprogram.sh&
```

nohup returns the pid of the running process.

Manage running processes

Each process has a unique identifier (pid). This number is what the user uses to manage each process (you can also use the process name). There are several commands to check the status of a running process. Take a quick look at these commands.

ps

The most common is the ps command. The default ps output is a simple list of running processes in the current Terminal, as follows, the first column contains the pid.

```
alan@workstation:~ $ps pid tty time cmd 23989 pts/0 00:00:00 bash 24148 pts/0 00
```

For example, you would like to see the nginx process started earlier. To do this, you need to tell ps to show yourself all running processes (-e) and a complete list (-f).

```
alan@workstation:~ $ps -ef uid pid ppid c stime tty time cmd root 1 0 0 aug18?00
```

You can see the nginx process in the output of the ps command above. This command shows almost 300 lines, but has been shortened in this example. As you can imagine, trying to handle 300 lines of progress information is a bit difficult. You can convert this output to grep and filter it to display only nginx.

```
alan@workstation:~ $ps -efgrep nginx alan 20520 1454 0 10:39?00:00:00 nginx:maste
```

You can quickly see that the pid of nginx is **20520** and **20521.**

pgrep

The pgrep command simplifies the problems encountered by calling grep alone.

```
alan@workstation:~ $pgrep nginx 20520 20521
```

Assuming you are in a hosted environment, many users are running several different versions of nginx. You can use the **-u** option to exclude others from output.

```
alan@workstation:~ $pgrep -u alan nginx 20520 20521
```

pidof

Another useful command is pidof. This will check the pid of a particular binary, even if another process with the same name is running. To build an example, I copied my nginx into the second directory and started with the corresponding path prefix. In fact, this example could be in a different location, such as folders owned by different users. If running two versions of nginx, the pidof output shows all processes.

```
alan@workstation:~ $ps -efgrep nginx alan 20881 1454 0 11:18?00:00:00 nginx:maste
```

Using grep or pgrep will display the pid number, but you may not know which version it is.

```
alan@workstation:~ $pgrep nginx 20881 20882 20895 20896
```

The pidof command can be used to identify the pid of a particular version of nginx.

```
alan@workstation:~ $pidof/home/alan/web/prod/nginxsec/sbin/nginx 20882 20881 alan
```

top

The top command has a long history, which is useful to see the details of running processes and to quickly identify problems like memory consumption. Its default view is shown below.

```
top-11:56:28 up 1 day, 13:37, 1 user, load average:0.09, 0.04, 0.03 tasks:292 to
```

You can change the update interval by entering the **s** and preferred update seconds. To easily track sample nginx progress, for example, you can use the **-p** option and switch pid to call top. This output is much cleaner.

```
alan@workstation:~ $top -p20881 -p20882 -p20895 -p20896 tasks:4 total, 0 running,
```

During process management, especially at the end of a process, it is important to correctly identify the pid. Also, if top is used in this way, whenever one of these processes stops or a new one starts, top needs to be notified of new processes.

End the process

kill

Interestingly, there is no stop order. In Linux, there is only the kill command. The kill command is used to send a signal to a process. The most common signals are **"sigterm"** or **"sigkill".** However, there is more to it than that. Following are some examples, the full list can be displayed with **kill -l** .

```
1) sighup 2) sigint 3) sigquit 4) sigill 5) sigtrap 6) sigabrt 7) sigbus 8) sigf
```

Note the 9th signal is **sigkill.** Usually you would send something like **kill -9 20896** . The default signal is **15 - sigterm** . Remember that many applications have their own stop methods. Nginx uses the **-s** option to switch signals, such as stop or reload. In general, most people prefer to use the application-specific method to stop working, however, I will demonstrate using the kill command to stop the nginx 20896 process, then use pgrep to verify receive that it has stopped. Pid 20896 no longer appears.

```
alan@workstation:~ $kill -9 20896 alan@workstation:~ $pgrep nginx 20881 20882 208
```

pkill

The pkill command is similar to pgrep in that it can search by name, which means you have to be very careful when using pkill. In this example nginx, if you only want to stop one instance of nginx, you probably won't choose to use pkill. You can either pass the nginx -s stop option to a specific instance to get rid of it, or use grep to filter the entire ps output.

```
/home/alan/web/prod/nginx/sbin/nginx -s stop /home/alan/web/prod/nginxsec/sbin/ng
```

If you want to use pkill, you can include the **-f** option to allow pkill to filter the entire command line argument. This of course also applies to pgrep. So, when executing **pkill -f** , you can use **pgrep -a first** . Confirm it.

```
alan@workstation:~ $pgrep -a nginx 20881 nginx:master process ./nginx -p/home/ala
```

You can also use **pgrep -f** to narrow your results. **pkill** stops processing with the same parameters.

```
alan@workstation:~ $pgrep -f nginxsec 20881 alan@workstation:~ $pkill -f nginxsec
```

The important thing to remember about pgrep (especially pkill) is that you should always make sure that your search results are accurate. This way you won't inadvertently compromise the inaccuracy.

Most of these commands have multiple options, so you should read the **man** page for each command. Although most of these commands exist on platforms like linux, solaris, and bsd, there are some differences. When working on the command line or writing scripts, always test them and be ready to fix them when needed.