A beginner’s introduction to the GNU/Linux command line, Part II

A tutorial for novices that discusses processes and process related commands including ps, top, grep, fuser, more, jobs and kill.

our GNU/Linux computer is an amazing machine. It can display images. It can run programs. It can perform dozens of functions all at the same time. How can you keep track of all this activity? By monitoring the *processes* that your computer runs, and one of the best ways to monitor and control processes is by using the command line.

Long ago computers were like calculators—they did one thing at a time—but today’s computers can multi-task doing hundreds of things at the same time. The different tasks share processor time. This is why you can search the web while writing a paper and listening to music all on the same computer at the same time.

What do you do when you can’t make your process stop no matter how many times you click that little x in the corner?

Each of these different things happening on the computer is called a *“process”*, and your computer takes turns letting all of the processes run a little bit at a time. The computer works so fast that you usually don’t even notice.

But occasionally, a process stops responding to you. What do you do when you can’t make your process stop no matter how many times you click that little x in the corner? You try using the keyboard commands Esc and Ctrl + C. You even try Ctrl + Q, Ctrl + X, and Ctrl + Z; but they don’t work. What do you do next?

If you can get into a terminal, you can kill any process

Well, if you can get into a terminal, you can kill any process. The command is called kill. The command name is easy enough to remember, but your problem is knowing what to kill. Each process has a different process identification number (PID) and you must use that number to kill it.

To see a list the processes currently running on your system, you can use the command ps. If I open a terminal program and type ps now, I will see this:

```
rosalyn@onizuka:~$ ps
PID TTY          TIME CMD
14036 pts/2    00:00:00 bash
14040 pts/2    00:00:00 ps
```

(The prompt on my system says rosalyn@onizuka:~$. Needless to say, your computer will say something different. When I show an example like this, only type the words after the dollar sign. The computer’s response will be shown on the following lines.)

The computer responds telling me what processes I am currently running in this terminal window. First, I am running the terminal window itself using the BASH shell. Then I am running the command ps. That’s not very informative. Is it?

I would really like the computer to show me all of the processes running on my system. To find that, I must use options (letters typed after the command that modify its function). In order to see all of the processes running on my system, I use the ps aux command:

```
rosalyn@onizuka:~$ ps aux
```

- ps reports a snapshot of the current processes running on your system.
- a tells the computer to show all processes
- u tells the user name of who owns the process, and
- x lists processes that were not started in a terminal.

This command will return a long list of processes that scrolls off my terminal screen. Somewhere in that list is the process ID that I am looking for. If I am using a terminal with a scroll bar, I can just scroll back up and read it all to find the process. Personally, however, I prefer to use a different command to get the process ID. The command called top.

Tasks and processes

What's the difference between a task and a process? Nobody knows. For the most part people use the word task when they are talking about sending things to the processor to avoid saying things like, "the processor processes the process".

Well, actually, there is a difference between tasks and processes. The process that starts your computer is called *init*. It is listed by the process ID 1. This process is also called Task [0]. For the most part, however, just consider tasks and processes to be the same thing.

Top lists the processes using the most time on the computer's processor. It prints only what will fit in your terminal window, and it refreshes every few seconds to give you an up to date picture of your system. If your program has stalled, then it is probably taking up lots of processor time, and it will be right near the top of the list. Typing top on my system shows me this:

```
top - 00:15:36 up 10:53,  1 user,  load average: 0.79, 0.69, 0.58
Tasks:  86 total,   3 running,  83 sleeping,   0 stopped,   0 zombie
Cpu(s): 29.9% us,  6.6% sy,  0.0% ni, 62.8% id,  0.0% wa,  0.3% hi,  0.3% si
Mem:    484292k total,   361820k used,   122472k free,    25596k buffers
Swap:  1421712k total,        0k used,  1421712k free,   169384k cached

PID   USER      PR  NI  VIRT  RES  SHR S %CPU %MEM    TIME+  COMMAND
13247 rosalyn   16   0  155m  67m  46m R 26.3 14.2  11:58.69 seamonkey-bin
3363 root      15   0 93448  50m  44m S  9.0 10.6   6:05.50 XFree86
14032 rosalyn   15   0 27320  14m  25m S  1.0  3.0   0:02.04 konsole
13227 rosalyn   15   0 14168 9928  11m S  0.7  2.1   0:09.65 gvim
3094 gkrellmd  15   0 18824 1340 2240 S  0.3  0.3   2:24.32 gkrellmd
13207 rosalyn   15   0 26532  14m  23m S  0.3  3.1   0:01.69 kdeinit
14196 rosalyn   16   0  2104 1084 1888 R  0.3  0.2   0:00.11 top
1 root      16   0  1516  520 1364 S  0.0  0.1   0:00.58 init
```

If you lengthen the terminal window, it will show more data. Now, suppose the program "*seamonkey*" was frozen. I could kill it by typing the command kill followed by seamonkey's process ID 13247.

Of course, to do this I need to get a prompt again. Top will keep refreshing in the terminal window until you tell it to stop. Both the letter q for quit and typing the control button and c at the same time (Ctrl + C) will stop the program and give you back your prompt.

To kill seamonkey, I can type:

```
rosalyn@onizuka:~$ kill 13247
```

This should stop the process causing the seamonkey window to close, but sometimes it doesn't. What do I do if seamonkey still won't close?

Well, kill also has options. The basic command tries to be nice. It says, "Could you please stop now?" But if the process just won't die when you ask it nicely, then use the option -9. The command kill -9 says "take no prisoners!"

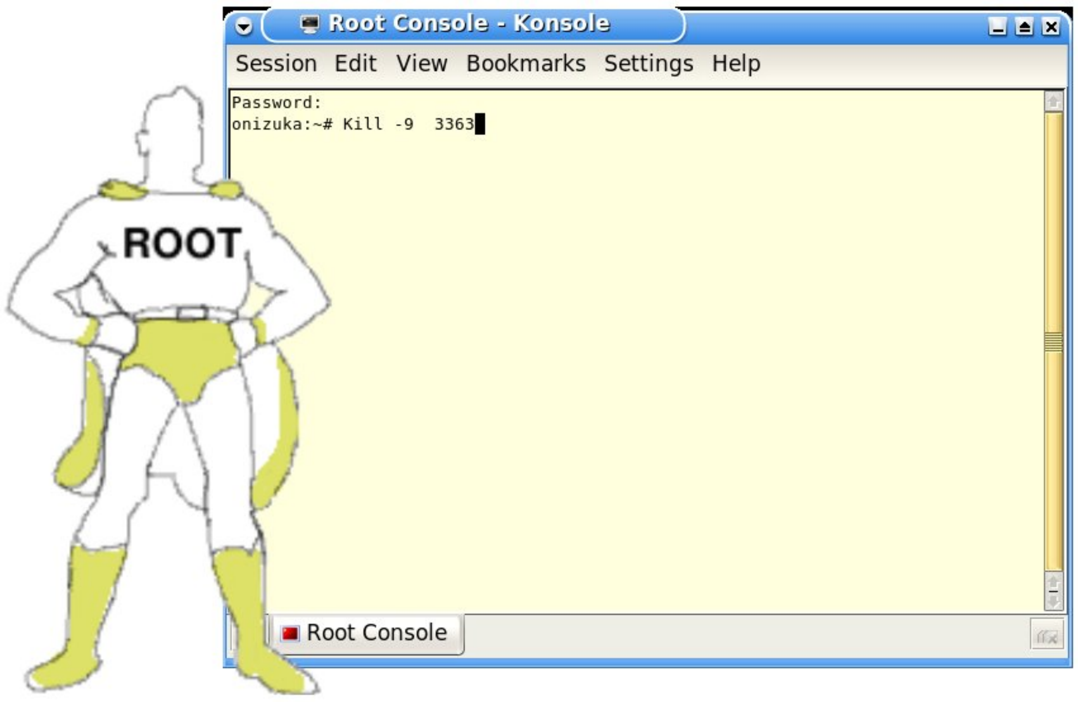
So if kill alone doesn't work, I can type:

```
rosalyn@onizuka:~$ kill -9 13247
```

That almost always will terminate the process.

Oh, by the way, you can only kill your own processes as a user. This protects you from accidentally shutting off someone else's programs. If you are logged in as a *superuser*, however, you can stop another person's programs.

Make sure to kill the right process though. In the listing above, you may notice that right below seamonkey is a listing for a process called XFree86. XFree86 is my window manager. Killing that would close all of my windows including the terminal that I am writing in now, so be cautious with kill.



```
Password:
onizuka:~# Kill -9  3363
```

The user Root is no normal user!

Users and superusers

GNU/Linux is designed to keep you from doing things that would damage your system. Each person is given their own account. All of a user's files are labeled as their own and other people usually can't change them. It is the same way for processes. Processes are labeled as belonging to someone, and you can only start and stop processes that belong to you.

But some processes belong to no one, and sometimes processes belonging to someone else get out of control. To affect those processes you have to become a *superuser*. A superuser, like a superhero, has amazing powers that mere mortals do not. They can start or stop any process; restrict access to files, or free them; add and remove users; and any of a number of incredible things. A superuser can also completely trash your system, so remember "with great power comes great responsibility".

A superuser, like a superhero, has amazing powers that mere mortals do not

Since being a superuser is so dangerous, most people only become one when they must. This is one of the major reasons that people use the command line. They login to a terminal as a superuser, and they logout when they have finished the task that they came to do.

On a GNU/Linux system, the superuser is called “root”. When you run top you will often find processes that are owned by root, and only root can kill those processes. To become root you must switch user using the command su. When you type su root it will prompt you for a password.

```
rosalyn@onizuka:~$ su root
Password:
```

You had to set a root password when you installed the Linux kernel, so the person who installed your system should have the root password. When you have the root password, you have ABSOLUTE CONTROL over the system (*diabolical laughter*), so it should not be given out casually.

If the person guarding the password refuses to give you this power, and you are tired of asking them to do something for you such as install new packages in a Debian system using apt-get install, they can give you partial power by using the command sudo. It means “switch user and do”.

Slay

There is a program called slay that will kill all of the processes used by another person. You can install it on your computer and access it from the command line. I use it when my son leaves several process-hogging games running in the background.

As a superuser, just type slay and a user name. The user’s processes will be terminated. Needless to say, doing this while someone is working on the computer is a way to lose a few friends. So, like all of these commands, caution is required as well as tact.

If you’d like to see what it feels like to get slayed, save any unsaved documents and then type slay and your user name. The effect is quite catastrophic and upsetting. Don’t say I didn’t warn you.

Using grep

Do you remember how I said that ps will give you a list of processes? And, lazy person that I am, I didn’t want to take the time to read them all. If you want to find a particular process in a long list, you can search the list using the command grep.

Grep is one of those tools that “oldetyme” programmers couldn’t survive without

When I type whatis grep on the command line it returns the following:

```
rosalyn@onizuka:~$ whatis grep
grep (1)            - print lines matching a pattern
```

Of course there are other ways for me to find a specific process. There are options to ps (type ps --help for the options.) The command ps -C, for example, lets you search by command name. There is even a different command called pstree that makes a layered diagram of all of the currently running processes. But let us imagine for now that we are option-challenged.

grep is one of those tools that “oldetyme” programmers couldn’t survive without. It searches a file for a string and prints it on the screen. You can write the data from the ps command into a file, and then search it with grep. Do this by using the greater-than arrow “>” to send the data to a file. Type the following:

```
rosalyn@onizuka:~$ ps aux > pslog.txt
```

This writes the output of the command ps aux to a file called *pslog.txt*.

Reading files on the command line

You can read a file on the command line by using the command cat. Typing cat pslog.txt will write the contents of the file to the screen, but it scrolls the entire file across the screen without stopping. If you want to read a file one page at a time, you can use the command more which will write only what fits on your screen. Press the space bar to scroll to the next page. Older versions of more only scroll text in one direction, down. Because of this, another program was written that can use the arrow keys to scroll up and down within a file. That program is called less. (Programmers have a weird sense of humor.)

Now that you have the file, you can search in it. Let's say you're looking for a program called *abiword*. You should type:

```
rosalyn@onizuka:~$ grep 'abiword' pslog.txt
```

Doing that returns the string:

```
rosalyn  13386  0.1  2.6 23264 12632 pts/3   S    08:16   0:00 /usr/bin/abiword
```

Now you've found Abiword's process ID. In my example it is 13386.

But in my mind's eye I can hear some veteran command line user out there scoffing. "Ha!" he says, "I could do all of that in one line." It is true that there are usually many ways to do the same thing when you are using the command line. I could simply type:

```
rosalyn@onizuka:~$ ps aux | grep "abiword"
```

The vertical line between the two commands is called a *pipe*. It is found above the backslash on the keyboard, and it is used to send the output of a process to another process just as the greater than arrow > is used to send the output of a process to a file. This command pipes the output of ps directly to grep and then prints it on the screen.

```
rosalyn@onizuka:~$ ps aux | grep "abiword"
rosalyn  13386  4.2  2.6 23264 12632 pts/3   S    08:16   0:00 /usr/bin/abiword
rosalyn  13396  0.0  0.1  1848  608 pts/2    R+   08:17   0:00 grep abiword
rosalyn@onizuka:~$
```

The first line shows the command you searched for and its PID 13386. The second line shows the ID for the command grep that you just ran to get this data. Pipes and arrows are really useful shorthands, and you can make quite complex commands that all fit on one line, if that makes you happy.

grep is a very useful tool. To see the options, try typing grep --help on the command line. Some of the more useful options are grep -r which will search recursively, meaning that it will search not only the files in a directory, but the files in all of the directories below it. grep -C which stands for *context* prints a number of lines that you specify above and below a line searched. So, for example, when I type:

```
rosalyn@onizuka:~$ grep -C3 'not to be' Hamlet.txt
```

the computer returns:

```
Enter Hamlet.

Ham. To be, or not to be, that is the Question:
Whether 'tis Nobler in the minde to suffer
```

The Slings and Arrowes of outragious Fortune,
Or to take Armes against a Sea of troubles,

In the example above, two of the three lines above the searched for line are blank. Cool isn't it?

Virtual terminals and jobs

In many versions of GNU/Linux, they have things called "virtual terminals". A virtual terminal allows you to simulate the experience of being on an old fashioned monochrome terminal. The kind they had before all this new-fangled window stuff was around. This is a necessary safety blanket for old programmers who miss the simplicity of it all. The great thing about having these virtual terminals is that if your graphical user interface hangs, you can switch to a virtual terminal and kill the hung window system. It is just another process after all.

How do you run a program from a terminal?

To get to a virtual terminal, you need to type at the same time the control key, the Alt key, and the F1 key. On my system I have six virtual terminals that I get to by typing Ctrl + Alt + F1 to Ctrl + Alt + F6. Pressing Ctrl + Alt + F7 returns me to the window system. This won't always work though. My other computer has no virtual terminals at all, so you will just have to try it and see.

What if you want to start a process and it isn't in your menu? How do you run a program from a terminal?

If you've just downloaded a program, and you are in the same directory as it, you can type ./filenameoftheprogram and it will run. If you don't know where the program is that you want to run, you can find its location by typing whereis. So if Abiword wasn't on my menu, but I knew that it was on my system and I wanted to run it, I could type:

```
rosalyn@onizuka:~$ whereis abiword
abiword: /usr/bin/abiword /usr/share/man/man1/abiword.1.gz
```

The computer gives me two locations for Abiword. The first listing is the actual program. The second listing is for the program's manual pages. Simply typing /usr/bin/abiword at the command line will start the program, but here is a case where a terminal is like a calculator. The prompt will vanish from the terminal, and won't return until you have closed the program.

To run a process from a terminal and still get the prompt back, you have to run it in the background. to do that put an ampersand at the end of the command.

```
rosalyn@onizuka:~$ /usr/bin/abiword&
[1] 12620
rosalyn@onizuka:~$
```

It gives me a job number and then returns the prompt so I can do something else. The second number on the line tells me the process ID, so I could kill the program if I so desired by typing:

```
rosalyn@onizuka:~$ kill 12620 [1]+ Terminated /usr/bin/abiword rosalyn@onizuka:~$
```

If I close the terminal that I started it in, Abiword will close, and I will really wish that I had saved my data

When you run a program from a terminal it is called a *job* and that terminal will number the jobs it is running sequentially. If I restart Abiword, and then type the command jobs, it returns the following:

```
rosalyn@onizuka:~$ jobs [1]+ Running /usr/bin/abiword & rosalyn@onizuka:~$
```

So I don't have to remember the long process ID. If I want to end this job, I just call it into the foreground with the command fg and the job number.

```
rosalyn@onizuka:~$ fg 1 /usr/bin/abiword
```

Then I can close it using the keyboard command Ctrl + C. The command fg brings a background program to the foreground, and the command bg runs a stopped job in the background. fg and bg are the terminal equivalent of tabbed browsing. This kind of thing was really hot stuff before multiple windows, but they are rarely used today. Heck, even multiple virtual terminals means that you can run processes in different terminals at the same time, so why bother?

If you have lots of small jobs that you want to oversee. For example, if you were running ten or twelve mathematical calculation programs, you could start each of them and then leave them in an open terminal while you go off to do something else. You could check their status by typing jobs from time to time, or even run top in the window for constant updates on their status.

One thing that you must remember about jobs is that they are tied to the terminal. Closing the terminal closes all of the jobs that you start in it. Therefore if I am typing a document in Abiword and I close the terminal that I started it in, Abiword will close, and I will really wish that I had saved my data.

Fuser and devices

Now, sometimes a process is halted because it is waiting for a device to finish. *Devices* are hardware attached to a computer such as a CDROM or a printer. Even the hard drive is a device. You can use the command fuser to see what process is using a device.

If I am playing music using the Kaffeine media player, then the device cdrom should be busy. If I type fuser /dev/cdrom it will give me the process ID being used by CDROM.

```
rosalyn@onizuka:~$ fuser /dev/cdrom /dev/cdrom: 15562 rosalyn@onizuka:~$
```

I can ask ps what program owns this PID by typing ps and the PID:

```
rosalyn@onizuka:~$ ps 15562 PID TTY STAT TIME COMMAND 15562 ? S 0:02 kaffeine rosalyn@onizuka:~$
```

Then I can kill this program using this program ID.

```
rosalyn@onizuka:~$ kill 15562
```

And yes, by looking at the list of options, I could learn how to find who has control of the device and kill it all on one line.

```
rosalyn@onizuka:~$ fuser -k /dev/cdrom /dev/cdrom: 15741
```

What is all this business about /dev/cdrom? In Unix-like operating systems, a device is referred to as if it were in a directory. Your cdrom may not be called /dev/cdrom like mine is. It may be /dev/cdrom0 or /dev/dvd. Since every system has different hardware, each /dev has a unique collection of files. But devices are a complete topic on their own.

Take control of your processes with the command line, and never be afraid of your software again

Processes are how your computer runs. Understanding processes helps you understand your computer, so open a terminal and type top. Leave it running and see what your system is doing from moment to moment. If you are curious about what a command does, type whatis and the command name. Remember, you run the computer it doesn’t run you. Take control of your processes with the command line, and never be afraid of your software again.

License

Biography

Rosalyn Hunter: Rosalyn Hunter has been on the internet since before the web was created. Born into a family of instructors, she has made it her life's goal to teach others about the important things in life, such as how to type kill -9 when a process is dead. She lives in a little house on the prairie in the American West with her husband, her three beautiful children, a cat and a dog.
