Boot Camp for Wizards, Part Four

Jerry Peek

You probably already know you can examine process status with `ps`, but most users never embrace the full power of `ps`. I've been talking about Linux concepts that wizards should know, but books often gloss over, and `ps` is at the top of the list. Let's dig into processes using `ps` and some of its more useful options.

Some Basics

The `ps` utility lists some or all of your system's current processes. By default — when you just run `ps` — you'll only see processes owned by you (running under your UID). Whether `ps` shows some or all of the processes, for all users or only for you, depends on its configuration and on your system. For instance, `ps` on a shared Web host may prevent non-root users from seeing other users' processes — even when using the "all" (`-A`) option.

The output of **ps** can also vary from system to system and kernel to kernel. Running **ps** on a multiprocessor system can show which processor a job is running on.

The output of `ps` can also vary from system to system and kernel to kernel. Running `ps` on a multiprocessor system, for instance, can show which processor a job is running on. (To find out how many processors your system has and their characteristics, try running `less /proc/cpuinfo`. We'll cover more of the */proc* filesystem in a later column.)

To find out more about your `ps`, read its man page and compare its output on various systems. The GNU version understands three different option syntaxes and has the usual long list of GNU features. Because the BSD (Berkeley) options (which don't start with a dash) give different output than the more-familiar Unix-style options (which do start with a dash), let's focus on them. Simple `ps` output can look like this:

```
$ ps x
  PID TTY      STAT   TIME COMMAND
10445 ?        S       0:00 sshd: jpeek@pts/0
10450 pts/0    Ss+    0:00 -bash
10499 ?        S       0:00 sshd: jpeek@pts/1
10501 pts/1    Ss     0:00 -bash
10508 pts/1    R+     0:00 ps x
```

By default, `ps` shows only processes on the current *tty* (typically, processes running in the terminal window where you're running `ps`). The `x` option used here tells `ps` to show processes owned by the user running `ps` on all ttys (all terminal devices) and all non-ttys too.

A `?` (question mark) in the `TTY` column shows that a process doesn't have a controlling tty — so it probably wasn't started from a terminal. This usually denotes processes being run as a `cron` job or other processes run by system daemons — for example, Web and mail servers.

To see a lot of processes without ttys, try running `ps ax`; the BSD-style `a` option shows all users' processes. In the example above, the process with PID 10445 is the `sshd` process that's spawned for an incoming SSH connection. The `COMMAND` column shows that it's for the user *jpeek* and it's using the tty *pts/0* — actually, */dev/pts/0*. (A process can manipulate its `COMMAND` information; `sshd` does a nice job of including useful text.)

Process 10450 is a shell running on the tty */dev/pts/0*. In fact, it was started by process 10445, though we can't be sure of that yet.

The `STAT` column tells a lot about the process state. `R` means the process is either running or runnable (can be run when the processor has time). `S` shows a process that's waiting for some event to finish — for instance, it's a shell waiting for a command to be input or for a child process to finish. The `PROCESS STATE CODES` section of the `ps` (1) man page tells more.

None of these processes have used much CPU time yet: all zero minutes and zero seconds (`0:00`). The times are rounded, of course. The processes have all used small amounts of CPU time.

Two last notes about this `ps x` listing. One is that the current user is logged in twice — using two ttys for two `bash` shells. Second is that `ps` is running on `pts/1` — which also tells you that the terminal window where you typed `ps x` is */dev/pts/1*. There's more trivia about ttys in the sidebar "A Terminal is a File."

Longer Output

The BSD-style `ps` option `l`(" long") gives more information. Let's look at some of it. (Try running `ps lx` yourself... and see its man page for explanation of the columns we've omitted.)

```
$ ps lx
 UID   PID  PPID PRI  NI   VSZ   RSS  CMD
1007 10445 10406   9   0  7528  2308  sshd:
```

```
1007 10450 10445    9    0   2532   1472   -bash
1007 10499 10474    9    0   7528   2308   sshd:
1007 10501 10499   15    0   3044   1748   -
bash
1007 11030 10501   14    0   1840    684   ps lx
```

The `UID` column shows which user ID the process is running under. These five processes are all owned by the same user: the one running `ps`.

The `PRI` column shows the current kernel scheduling priority for the process. The `NI` column shows the "niceness" level, which is typically set by a user running the `nice` or `renice` commands. If you increase a niceness of a process, the kernel typically gives it less priority — which tends to let other processes run more often (get more CPU time). In other words, increasing the niceness of a process is being "nice" to other users and other processes. If you're the superuser, you can decrease a process' niceness to ask the scheduler to give the process more CPU time. Non-root users can't decrease the process niceness. If your system is busy, you and other users should consider using `nice` and `renice` on any low-priority jobs. Don't "nice" an interactive process, though: it might not get enough CPU time to respond to what you type.

None of the processes in this example have been "niced"; they all have values of 0.

VSZ is the virtual memory, in kilobytes, reserved for the process. RSS is the resident set size, the amount of physical (non-swapped) memory that the process is currently using.

Notice that you can identify each process by its PID. All processes in this `ps lx` listing have the same PID as in the previous `ps x` listing except for the `ps` itself: it's a new instance, so it has a new PID. (PIDs recycle eventually from low numbers, but a process keeps the same unique PID until it terminates — even if that's days or months later.)

Parent and Child Processes

The PPID is the PID of the parent process — typically, the process that started this process and is waiting for it to finish. So, the first `bash` shell, PID 10450, was started by the `sshd` whose PID is 10445. And the `ps lx` was started by the `bash` with PID 10501, which itself was started by the `sshd` with PID 10499.

Where are the parents of the `sshd` processes? They aren't owned by UID 1007, so `ps` isn't showing them. You can see their listings by giving their PIDs to `ps` with the `-p` option. (We're purposely mixing Berkeley and Unix `ps` option styles here.) In the next example, you can see that both of the `sshd` processes are running as root (UID 0); both also have the same parent process; its PID is 31477. And that "grandparent" process has a PPID of 1:

```
$ ps l -p 10445,10474
 UID    PID   PPID TTY   COMMAND
   0 10445 31477 ?     sshd: jpeek
   0 10474 31477 ?     sshd: jpeek
$ ps l -p 31477
 UID    PID   PPID TTY   COMMAND
   0 31477      1 ?     /usr/sbin/sshd
```

Process 1 is `init`, the parent (or grandparent, or...) of all other process. `init` also becomes the parent of "orphaned" processes that have been "disowned" by other processes — for instance, by shells that exit while a background process is still running.

Tracking parent and child processes by their PID and PPID can be tedious. Try `ps -Hl` (uppercase "H", lowercase "l")

See Power, *pg. 52*

A TERMINAL IS A FILE

The first column in this Boot Camp series mentioned that Linux data is typically a stream of bytes. You can write that stream to basically any device — a printer, disk drive, or a FIFO, for instance — and the system handles any translation needed.

We saw that today's `ps x` command was running on */dev/pts/1*. This *pty* is a file like any other, so you can write text to it. Let's send a message to that terminal:

```
$ echo "Hi there $USER" > /dev/pts/1
```

If you run that command from */dev/pts/1*, its output will look the same as if you ran echo "`Hi there $USER`" without the redirection operator. Why? Because, by default, the shell's standard output goes to its own *tty* — and so does the standard output of echo. If you redirect the output of echo to */dev/ pts/1*, that doesn't change anything: the *stdout* of echo still goes to */dev/pts/1*.

If you run the command from another terminal, you'll see its output on whatever tty you redirect its output to — as long as you own the tty you're trying to write to, that is. Your system probably only allows other non-root processes to write to your tty if the process is a member of the group named *tty*. Check that by running ls -l on your terminal file:

```
$ ls -l /dev/pts/1
crw—w—— 1 jpeek tty ... /dev/pts/1
```

The ls-l output shows that this character device (note the leading c) can be read and written by the user *jpeek*, and written by other processes with the group *tty*. Other processes can't read or write the termina l — which is good for security. (Older Unix systems gave write permission on all terminals to all users. That allowed inter-terminal discussions with the ancient utility write, but it also allowed mischief. Users could deny permission to other users by running `mesg` n; the mesg utility controlled write permission for other users on the current tty.)

Power, *from pg. 19*

for a hierarchical long view. For instance, running `man` from a `bash` shell makes a child `man` process, which starts an `nroff` process and a `pager`. The `nroff` starts a `groff`, which starts `troff` and `grotty`:

```
$ ps -Hl
 PID  PPID  ... CMD
3481  3477  ... bash
5057  3481  ...   man
5065  5057  ...     nroff
5069  5065  ...       groff
5075  5069  ...         troff
5076  5069  ...         grotty
5066  5057  ...     pager
```

LISTING ONE: User-defined ps format in an alias

```
$ alias psid='ps- o" pid ppid ruid euid rgid \
  egid args"'
$ psid- p 962
PID PPID RUID EUID RGID EGID COMMAND
962 1 0 1000 0 100 /usr/sbin/famd- T 0
```

Various options, including `ps -F` (uppercase "F") and `ps u`, give the clock time when a process started and the name of the user instead of the UID. One way to see the percentage of CPU and memory being used by a process is with `ps u`.

User-defined Output Formats

As you can see, `ps` has a lot of output options and styles, and this column has only scratched the surface! Trying to remember what information you get from all of these can be tough. The best way to choose what you want may be to make yourself a shell alias or function that uses the format options `o`, `-o`, or `—format`. The Standard Format Specifiers section of the `ps` man page describes the output columns you can choose; you can also run `ps L` for a list.

For instance, *Listing One* shows an alias named *psid* that gives the PID, parent PID, as well as the real and effective UID and GID, along with the process name and its arguments. Running the alias on PID 962 shows a process that started life as root (real UID and GID of 0) but that now has a different effective UID and GID. This is the kind of specific information you can get with a custom `ps` output format.

Jerry Peek is a freelance writer and instructor who has used Unix and Linux for more than 25 years. He's happy to hear from readers: see http://www.jpeek.com/contact.html.