Processes and Process Monitoring

CS14 - S26

xargs

Converts standard input to command line arguments. Useful for pipelines where a command needs args instead of stdin, or behaves differently with args instead of stdin.

E.g.,

```
$ find -name foo.txt | cat   (fails)
$ find -name foo.txt | xargs cat   (works!)
```

Exercise: In a tree_nav exercise, use `find`, `grep`, and `xargs` to print the contents of all `.txt` files containing the capital letter “A”.

Processes

Process - A program that is executing on a system.

There are many ways to start, monitor, and manage processes.

In Linux, each running process is assigned a unique number, known as its process ID (PID). Many commands either list or take as arguments PIDs.

Additionally, each process is associated with a user, the user who started the process.

Daemon Processes

Processes that are run by the system to manage services, and that are not meant to be executed directly by a human user are called “daemons”.

Processes that are daemons will often have “daemon” or a “d” at the end of their names. E.g. `gnome-keyring-daemon`, `systemd`

Daemons typically belong to either the root user, or special users created just to run daemons.

Process monitoring - ps

`ps` - “Process Status”. Prints a snapshot of currently running processes.

Exercise: By default, `ps` only prints the processes running in the current terminal. Looking at the `man` page, use `ps` to print all running processes. Then, print all processes that are owned only by your user account.

Process monitoring - top and htop

`top` is a program the displays information about processes and system resource usage in real time. `htop` is a more recently developed alternative to `top` with better default coloring and graphics and more options. However `htop` may not be installed by default.

Exercise: Open `htop` and try scrolling with the arrow keys. Try searching for a process name using forward slash followed by a name. Press h to open the help page and look at more options.

Foreground and background process, and nohup

When you execute a process at the shell, it normally starts in “foreground” mode.

- The shell will be suspended until the process finishes.
- If the terminal running the shell closes before a shell child processes finishes, the child process will typically receive a “hangup” signal and stop.

You can run processes in the “background” by appending `&` to the end of a command.

- The process will start, and you can still enter shell commands.
- The PID of the background process is printed.
- Useful for some long running processes, e.g. `xclock &`

Foreground and background process, and nohup

nohup makes following commands ignore “hangup” signals.

- Allows shell child processes to continue when the shell process ends.
- E.g. `nohup xclock`

Combining nohup and & makes child processes return control to the shell, and allows child processes to run after the terminal is closed.

- E.g. `nohup xclock &`

Foreground and background process, and nohup

Ctrl-z is used to suspend a foreground process.

Suspending (temporarily) halts execution of a process, rather than just sending it to the background.

`fg` is used to either bring a background process to the foreground, or resume a suspended process.

Signals

Signals are one mechanism for processes to communicate with one another. Processes can send and receive many different kind of signals to cause different behaviors. See `man 7 signal` for information on POSIX standard signals.

Some signals you may have seen before:

- SIGINT
- SIGKILL
- SIGSEGV
- SIGCHLD
- SIGHUP

For more information on coding with and handling signals in programs, take CS31 and CS45.

kill and killall

`kill` sends a signal to a particular PID.

`killall` sends a signal to all processes with the same name.

- E.g. `killall -9 firefox` sends SIGKILL to every firefox process you have permissions for.

Use `kill -L` to see a list of signals by number.

Exercise: Open two terminals. In one terminal, try running some processes both in the foreground and background. In the other, try using both `kill` and `killall` to send SIGTERM to those processes.