csce215 — UNIX/Linux Fundamentals
Spring 2022 — Lecture Notes: Processes and jobs

This document contains slides from the lecture, formatted to be suitable for printing or individual reading, and with some supplemental explanations added. It is intended as a supplement to, rather than a replacement for, the lectures themselves — you should not expect the notes to be self-contained or complete on their own.

(7.1) *Last time*

Last time we learned about **finding things** using tools like `find`, `locate`, and `grep`, along with some details about using **regular expressions** within those tools.

Today, we'll cover three important but assorted topics:

- Running shell commands in the background using **job control**.
- Listing and terminating **processes**.
- Understanding and modifying **permissions** on files and directories.

(7.2) *Jobs*

A **job** is a pipeline of one or more programs launched from a shell command.

Here's an example of one job consisting of three programs:

```
$ find /usr/share | grep hello | head -n5
/usr/share/cowsay/cows/hellokitty.cow
/usr/share/cmake-3.16/Modules/IntelVSImplicitPath/hello.f
/usr/share/locale-langpack/en_AU/LC_MESSAGES/hello.mo
/usr/share/locale-langpack/en@quot/LC_MESSAGES/hello.mo
/usr/share/locale-langpack/en_CA/LC_MESSAGES/hello.mo
```

Each job always in one of three states:

- **Foreground**: Running, with control of the terminal. (This is the default.)
- **Background**: Running, but not able to read from the terminal.
- **Stopped**: Waiting to be resumed.

(7.3) *Listing jobs*

`jobs`
List all jobs within the current shell.

For example, during the lectures, a PDF viewer is running the background:

```
$ jobs
[1]+  Running                 evince build/slides-jobs.pdf &
```

(7.4) *Job control commands*

There are two commands to put jobs into the foreground or background:

To background a stopped job, use the `bg` command.

`bg`
Put the given job into the background.

To foreground a stopped or background job, use the `fg` command.

`fg`
Put the given job into the foreground.

(7.5) *Starting new jobs*

When starting a new job:

> Normally, new jobs start in the **foreground**.

But:

> Use a single **ampersand** `&` at the end of the command to start it in the **background** instead.

(7.6) *Stopping jobs*

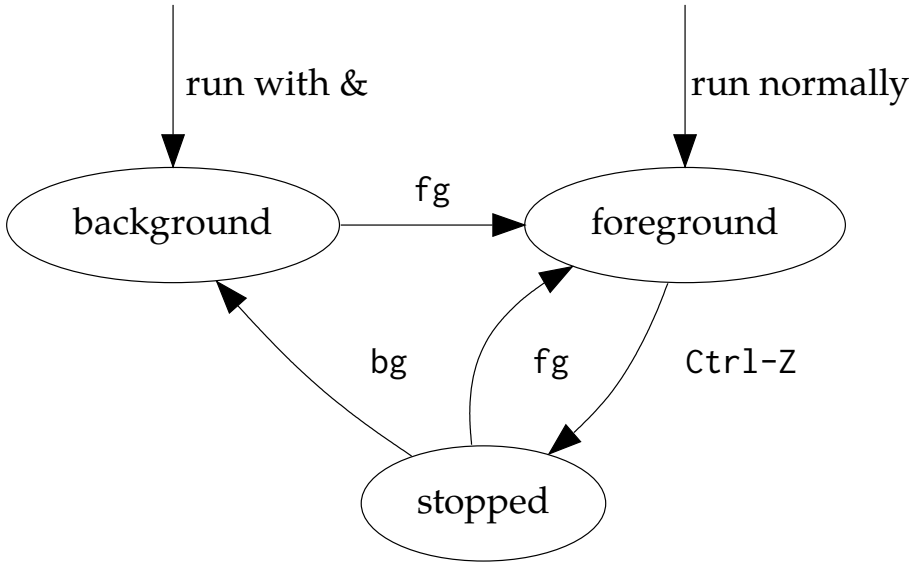
There is no *command* to stop the current job.

> (Why not? That would not make sense, since if there's a foreground job, there's no command prompt to type the command!)

Instead, to stop the current foreground job, use `Ctrl-Z`.

- The current job goes to a stopped state. It can be resumed later.
- The command prompt returns.

(7.7) *Job control summary*



(7.8) *Processes*

A Linux system generally has a number of **processes**, i.e. running programs, active at any time.

Example processes:

- Your shell.
- Any commands currently running or stopped.
- Your web browser (or possible just one tab).
- Your terminal emulator, PDF viewer, etc.
- Various servers, daemons, and other background processes.
- ...

In many systems, it is common to have a few hundred active processes, owned by the system itself or by one of the users.

Each of the *jobs* we can control with `fg`, `bg`, `Ctrl-Z`, etc. is made up of one or more processes.

(7.9) *Listing processes*

Use `ps` to list processes associated with the same user and the same terminal as `ps` itself.

`ps`

List processes.

`-A` List *all* processes.

`-f` Show additional columns.

```
$ ps
    PID TTY          TIME CMD
 228346 pts/2    00:00:00 bash
 236924 pts/2    00:00:00 python3
 236937 pts/2    00:00:00 bash
 236952 pts/2    00:00:00 bash
 236955 pts/2    00:00:00 ps
```

Use `ps -A` to list every process on the system.

```
$ ps -A | wc -l
363
$ ps -A | tail
 235947 ?        00:00:00 kworker/3:3
 236169 ?        00:00:00 kworker/5:0
 236467 ?        00:00:00 kworker/0:1-kacpi_notify
 236539 ?        00:00:00 kworker/u17:1
 236808 ?        00:00:00 tracker-store
 236924 pts/2    00:00:00 python3
 236937 pts/2    00:00:00 bash
 236952 pts/2    00:00:00 bash
 236958 pts/2    00:00:00 ps
 236959 pts/2    00:00:00 tail
```

(There are loads of other options for `ps`. Have a look at the `man` page.)

(7.10) *Monitoring processes*

Sometimes we want to keep an eye on the processes that are consuming the most resources.

`top`
Continually display processes.

This will continue updating until you terminate it with `Ctrl-C`.

(7.11) *Killing a process, politely*

If a process is misbehaving, you may need to kill it.

`kill`
Ask the process with the given `PID` to terminate. `-9` Force the process to end, even if it is not cooperating.

(7.12) *Killing a lot of processes*

Sometimes there may be several processes to kill all at once.

`killall`
Ask all processes running the specified command to terminate. `-9` Force the processes to end, even if they are not cooperating.

(7.13) *Permissions*

Every file and directory has a set of 9 **permission bits** that determine who is allowed to access that file and what they're allowed to do.

We've seen these permissions already!

Remember the first column of output from `ls -l`?

```
$ ls -l pictures
total 128
drwxrwxr-x 2 jokane jokane  4096 Dec  7  1941 blurry
-rw-rw-r-- 1 jokane jokane 77931 Dec  7  1941 photo1.jpg
-rw-rw-r-- 1 jokane jokane 41772 Dec  7  1941 photo3.jpg
```

(7.14) *Types of users*

Permissions can be set for each of three groups of users:

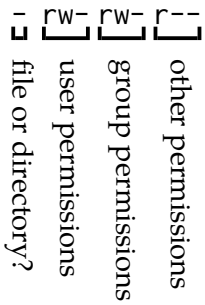
- The file's owner, i.e. the user.

  Who is the owner? See third column in `ls -l`.

- Members of the file's group.

  What group? See fourth column in `ls -l`. *Usually, this the user's 'primary group', which is named after the user.*

- Others, i.e. everyone else.



(7.15) *Types of access*

Permission can be granted for three types of access:

- Read

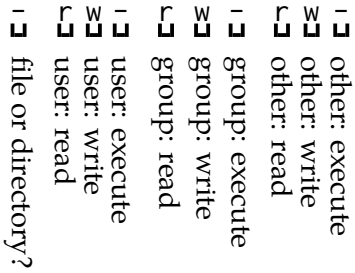
  May I view the contents of this file?

- Write

  May I modify the contents of this file?

- Execute

  May I execute this file as a program?



(7.16) *Permissions on directories*

For directories, the permission bits mean:

- Read

  May I see a list of files in the directory?

- Write

  May I create, delete, or rename files in the directory?

- Execute

  May I use `cd` *to 'go to' this directory?*[1]

(7.17) *Changing permissions*

A file's owner can change its permissions.

`chmod`

Change permissions for files and directories.

`u` set permissions for user/owner

`g` set permissions for group

`o` set permissions for others

`+` add permissions

`-` remove permissions

`r` set read permissions

`w` set write permissions

`x` set execute permissions

To make a program executable:

```
$ cat hi.sh
#!/bin/bash
echo Hello, world
$ ./hi.sh
/bin/bash: line 4: ./hi.sh: Permission denied
$ chmod u+x hi.sh
$ ./hi.sh
Hello, world
```

(7.18) *A practical example: Your web page*

If you put files in a subdirectory called

[1] More precisely: May I 'search' this directory?

within your home directory, with appropriate permissions, they'll be available on the web at

`https://cse.sc.edu/~username`

The web server process usually runs as a user called `www-data`. How can we allow that user to access the files?

Step 1: Allow access to the directory.

```
$ chmod -v o+rx ~/public_html
mode of '/home/jokane/public_html' retained as 0755 (rwxr-xr-x)
```

Step 2: Allow access to the individual files.

```
$ chmod -v o+r ~/public_html/index.html
mode of '/home/jokane/public_html/index.html' retained as 0644 (rw-r--r--)
```

Exercise: These commands work only for a single directory and single file, but don't work if there are subdirectories inside `public_html`. How could you set the permissions for `public_html` and all of its subdirectories? For all of the files in those subdirectories?[2]

[2]Hint: Do you know a way to `find` files or directories and execute a command for each one?

(7.19) *Sample final exam questions*

1. The purpose of the `fg` command is _____.

   A. to open the floodgates
   B. to show the number of free gates
   C. to send the given process into the State Fairgrounds
   D. to put the given job into the foreground

2. To change a file's permissions, giving read and execute permission to the user/owner, what option would be given to the `chmod` command?

   A. `o-rx`
   B. `o+rx`
   C. `u+rx`
   D. `u-rw`

3. Why is it important to sometimes use the `-9` flag with the kill command?

   A. It tells `kill` to end all processes associated with the current shell.
   B. It tells `kill` to find and kill the process using the most memory.
   C. It tells `kill` to end all processes in the system.
   D. It tells `kill` to force the process to end, even if it is not cooperating.

4. Which of these commands will end a process?

   A. `kill`
   B. `end`
   C. `stop`
   D. `terminate`

5. To change a file's permissions, removing write permission from all users, what option would be given to the `chmod` command?

   A. `g-w`
   B. `w-w`
   C. `a-w`
   D. `u-w`

6. Which of these commands is used to change permissions on files and directories?

   A. `modch`
   B. `allow`
   C. `perm`
   D. `chmod`

7. Which of these commands is used to send a job to the background?

   A. `ps`
   B. `bg`
   C. `fg`
   D. `jobs -bg`

8. Which of these commands shows the correct way to start a program as a background job?

A. `bg ./program.py`
B. `fg ./program.py`
C. `bg ./program.py &`
D. `./program.py &`

9. We need to stop the current foreground job! What should we type?

A. Ctrl-Y
B. Ctrl-Z
C. Ctrl-Q
D. Ctrl-X

10. Which of these commands will display a list of all processes on the system?

A. `ps -f`
B. `ps -A`
C. `ps -all`
D. `ps`

11. The purpose of the `jobs` command is to _____.

A. list all jobs within the current shell
B. create new jobs within the current shell
C. list only stopped jobs within the current shell
D. list only running jobs within the current shell