CH III Processes

Section Objectives

- Unix is multi-user, multi-process OS
- Shell features to control jobs
- Unix utilities to manage jobs
  - crontab
  - at
  - batch

Terminology

- Process is a program in execution
  - process is created every time you run a command
  - each process has a unique process id
  - processes are removed from the system when the command finishes its execution

- job is a unit of work
  - consists of the commands specified in a single command line
  - A single job may involve several processes, each consisting of an executable program

Job Control Terminology

- Foreground job:
  - a job that has our immediate attention
  - user has to wait for job to complete
- Background job:
  - a job that the user does not wait for
  - it runs independently of user interaction
- Unix shells allow users to:
  - make jobs execute in the background,
  - move jobs from foreground to background,
  - determine their status, and terminate them

Background Jobs

- How do we decide which jobs to place in the background?
  - jobs that are run non-interactively
  - jobs that do not require user input

<u>Examples:</u>

- searching the file system for particular kinds of files
- solving complex equations
- compiling long programs
- backing up the file system

Background Jobs

- to execute command in the background, put

  &

  after it

<u>Example:</u>

```
gedit&
```

Managing jobs

- display jobs
  - command “jobs” lists your active jobs
  - each job has job number
  - job number with “%” is used to refer to job
- send job to background
  - bg
- move job to foreground
  - fg

Ps vs top

- ps: Displays the real time state of the process.
- top: Viewing regularly updated processes running.
- Top allows you display of process statistics continuously until stopped vs. ps which gives you a single snapshot.

To show the processes for the current running shell run `ps`. If nothing else is running this will return information on the shell process being run and the `ps` command that is being run.

```
ps
PID  TTY          TIME CMD
5763 pts/3    00:00:00 zsh
8534 pts/3    00:00:00 ps
```

The result contains four columns of information.

- `PID` - the number of the process
- `TTY` - the name of the console that the user is logged into
- `TIME` - the amount of CPU in minutes and seconds that the process has been running
- `CMD` - the name of the command that launched the process

To list all processes on a system use the `-e` option.

```
ps -e
PID TTY          TIME CMD
  1 ?        00:00:01 systemd
  2 ?        00:00:00 kthreadd
  3 ?        00:00:00 ksoftirqd/0
....
```

This option can be combined with the `-f` and `-F` options to provide more information on processes. The `-f` option offers full-format listing.

```
ps -f
UID        PID  PPID  C STIME TTY          TIME CMD
root         1     0  0 19:58 ?        00:00:01 /sbin/init
root         2     0  0 19:58 ?        00:00:00 [kthreadd]
root         3     2  0 19:58 ?        00:00:00 [ksoftirqd/0]
...
```

Another commonly used syntax to achieve seeing every process on the system using BSD syntax is `ps aux`.

```
ps aux
USER         PID %CPU %MEM    VSZ   RSS TTY      STAT START   TIM
root           1  0.0  0.0  53120  6368 ?        Ss   Sep19   0:0
root           2  0.0  0.0      0     0 ?        S    Sep19   0:0
...
```

To list all processes by user use the `-u` option. This supports the user ID or name.

```
ps -u george
 PID TTY          TIME CMD
1053 ?        00:00:00 systemd
1062 ?        00:00:00 (sd-pam)
1074 tty1     00:00:00 zsh
...
```

How to list all processes for a group

To list all processes by group use the `-g` option. This supports the group ID or name.

```
ps -g users
 PID TTY          TIME CMD
 997 ?        00:00:00 login
1053 ?        00:00:00 systemd
1062 ?        00:00:00 (sd-pam)
...
```

How to list all processes by process number

To list all processes by process number use the `-p` option. This selects the processes whose numbers match the list provided to the `-p` option.

```
ps -p 12608 3995
  PID TTY      STAT   TIME COMMAND
 3995 ?        Ss     0:00 st
12608 pts/2    S+     0:04 vim content/post/unix-ps.md
```

How to list all processes by executable name

To list all processes by executable name use the `-C` option. This selects the processes whose executables match the list of executables given to the `-C` option.

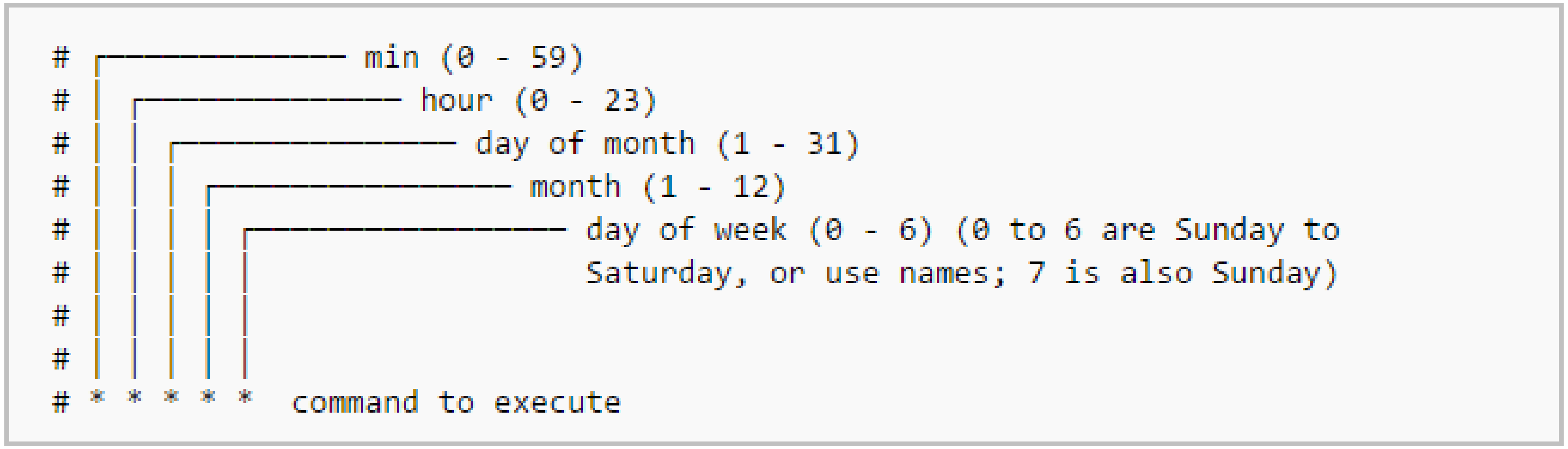
```
ps -C tmux
 PID TTY          TIME CMD
5733 pts/0    00:00:00 tmux
5735 ?        00:00:06 tmux
```

Kill

- killall terminates running processes based on name
- kill terminates processes based on Process ID number (PID)
- kill and killall can send a specified signal to a specified processes or process groups.
- When used without a signal both tools will send -15 (-TERM).
- The most commonly used signals are:
  - 1 (-HUP): to restart a process and reload configuration file.
  - 9 (-KILL): to kill a process.
  - 15 (-TERM): to gracefully stop a process.
- Signals can be specified in three different ways:
  - using number (e.g., -1)
  - with the “SIG” prefix (e.g., -SIGHUP)
    - without the “SIG” prefix (e.g., -HUP).
- Use the -l option to list all available signals:

crontab

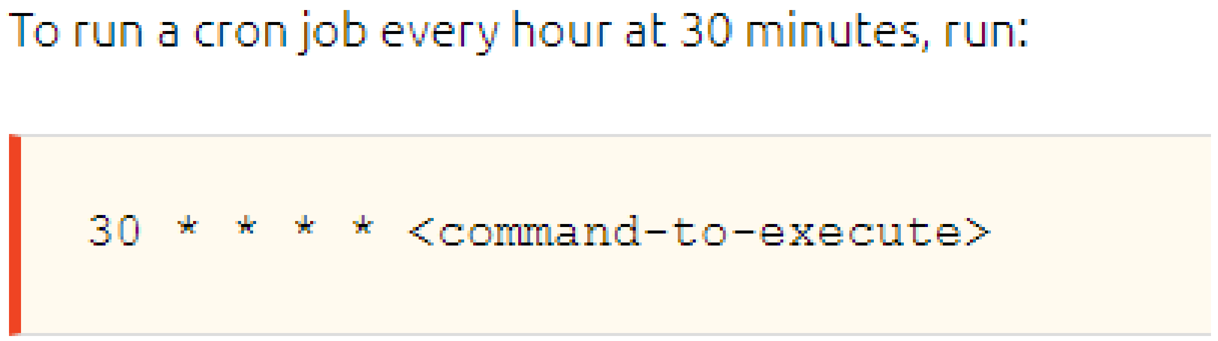
- The cron daemon is a long-running process that executes commands at specific dates and times.
- You can use this to schedule activities, either as one-time events or as recurring tasks.
- To schedule one-time only tasks with cron, use the at or batch command



```
# ┌───────────── min (0 - 59)
# │ ┌────────────── hour (0 - 23)
# │ │ ┌─────────────── day of month (1 - 31)
# │ │ │ ┌──────────────── month (1 - 12)
# │ │ │ │ ┌───────────────── day of week (0 - 6) (0 to 6 are Sunday to
# │ │ │ │ │                  Saturday, or use names; 7 is also Sunday)
# │ │ │ │ │
# │ │ │ │ │
# * * * * *  command to execute
```

crontab

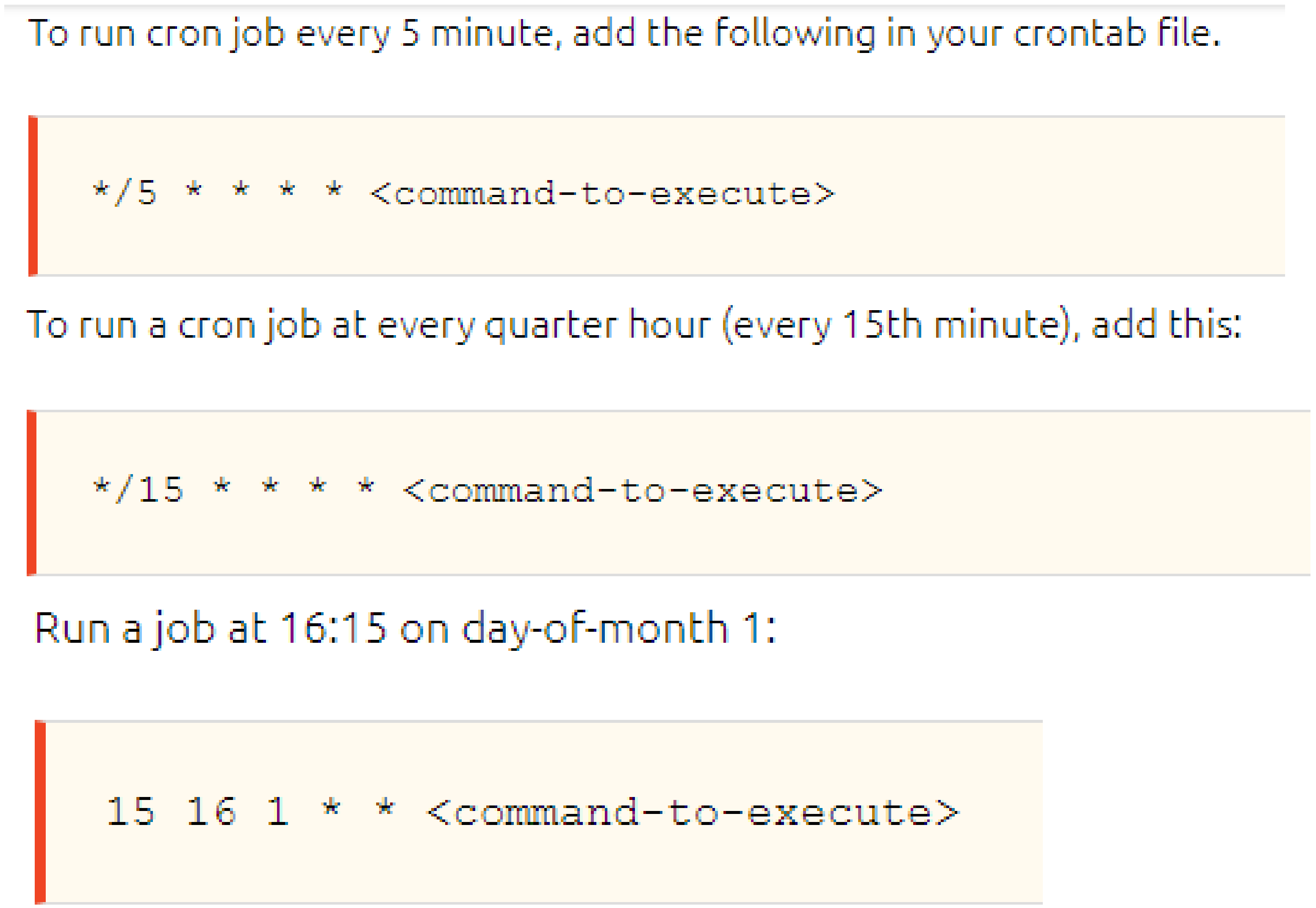
- To display the contents of the **crontab** file of the currently logged in user: crontab –l
- To edit the current user’s cron jobs, do:
  crontab –e



To run a cron job every hour at 30 minutes, run:

```
30 * * * * <command-to-execute>
```

Crontab examples



To run cron job every 5 minute, add the following in your crontab file.

```
*/5 * * * * <command-to-execute>
```

To run a cron job at every quarter hour (every 15th minute), add this:

```
*/15 * * * * <command-to-execute>
```

Run a job at 16:15 on day-of-month 1:

```
15 16 1 * * <command-to-execute>
```

Crontab examples

Run a job at every quarter i.e on day-of-month 1 in every 3rd month:

```
0 0 1 */3 * <command-to-execute>
```

Run a job on a specific month at a specific time:

```
5 0 * 4 * <command-to-execute>
```

The job will start at 00:05 in April.

Run a job every 6 months:

```
0 0 1 */6 * <command-to-execute>
```

Crontab examples

Run a job every day at 3am:

```
0 3 * * * <command-to-execute>
```

Run a job every sunday:

```
0 0 * * SUN <command-to-execute>
```

Run a cron job every half hour:

```
*/30 * * * * <command-to-execute>
```

Run a job every day (It will run at 00:00):

```
0 0 * * * <command-to-execute>
```

Run a job every hour:

```
0 * * * * <command-to-execute>
```

Run a job every 2 hours:

```
0 */2 * * * <command-to-execute>
```

Run a job every sunday:

```
0 0 * * SUN <command-to-execute>
```

Or,

```
0 0 * * 0 <command-to-execute>
```

Crontab examples

Run a job on every day-of-week from Monday through Friday i.e every weekday:

```
0 0 * * 1-5 <command-to-execute>
```

The job will start at 00:00.

Run a job every month:

```
0 0 1 * * <command-to-execute>
```

You can also define multiple time intervals separated by commas. For example, the following cron job will run three times every hour, at minutes 0, 5 and 10:

```
0,5,10 * * * * <command-to-execute>
```

crontab command

<u>options:</u>

-e	to edit the control file
-l	to list the control file
-r	to remove the control file

- for superuser

-u	to edit another user’s control file

One Time Execution: at utility

- Use ‘at’ to run a command or list of commands at a later time
- Must specify on the command the time and date on which your command to be executed
- Do not have to be logged in when the commands are scheduled to run

Syntax:

```
   % at timeDate command
```

at utility

- Can give as much of date as desired
- If date/time has passed, command will run instantly
    - In case system was down when it was supposed to run

Examples:

```
% at 13:45 Wed
% at 01:45 pm Sep 18
% at 09:25 Sep 18 2010
% at 09:25 Sep 18, 2010
% at 11:00 pm tomorrow
% at teatime              # 4:00 pm
```